W3School PHP 教程

wizardforcel

Published with GitBook



目錄

W3School PHP教程

来源：PHP教程

整理：飞龙

PHP 基础

PHP 简介

PHP 脚本在服务器上执行。

您应当具备的基础知识

在继续学习之前，您需要对下面的知识有基本的了解：

- HTML
- CSS
- JavaScript

如果您希望首先学习这些项目，请在我们的 首页 访问这些教程。

什么是 PHP？

- PHP 是 "PHP Hypertext Preprocessor" 的首字母缩略词
- PHP 是一种被广泛使用的开源脚本语言
- PHP 脚本在服务器上执行
- PHP 没有成本，可供免费下载和使用

PHP 是一门令人惊叹的流行语言！

- 它强大到足以成为在网络上最大的博客系统的核心（WordPress）！
- 它深邃到足以运行最大的社交网络（facebook）！
- 而它的易用程度足以成为初学者的首选服务器端语言！

什么是 PHP 文件？

- PHP 文件能够包含文本、HTML、CSS 以及 PHP 代码
- PHP 代码在服务器上执行，而结果以纯文本返回浏览器
- PHP 文件的后缀是 ".php"

PHP 能够做什么？

- PHP 能够生成动态页面内容
- PHP 能够创建、打开、读取、写入、删除以及关闭服务器上的文件
- PHP 能够接收表单数据
- PHP 能够发送并取回 cookies
- PHP 能够添加、删除、修改数据库中的数据
- PHP 能够限制用户访问网站中的某些页面

- PHP 能够对数据进行加密

通过 PHP，您可以不受限于只输出 HTML。您还能够输出图像、PDF 文件、甚至 Flash 影片。您也可以输出任何文本，比如 XHTML 和 XML。

为什么使用 PHP？

- PHP 运行于各种平台（Windows, Linux, Unix, Mac OS X 等等）
- PHP 兼容几乎所有服务器（Apache, IIS 等等）
- PHP 支持多种数据库
- PHP 是免费的。请从官方 PHP 资源下载：www.php.net
- PHP 易于学习，并可高效地运行在服务器端

PHP 安装

我需要什么？

如需开始使用 PHP，您可以：

- 使用支持 PHP 和 MySQL 的 web 主机
- 在您的 PC 上安装 web 服务器，然后安装 PHP 和 MySQL。

使用支持 **PHP** 的 **Web** 主机

如果您的服务器支持 PHP，那么您无需做任何事情。

只要创建 .php 文件，然后上传到 web 目录中即可。服务器会自动对它们进行解析。

您无需编译或安装任何额外的工具。

因为 PHP 是免费的，大多数 web 主机都支持 PHP。

在您的 **PC** 上运行 **PHP**

不过如果您的服务器不支持 PHP，那么您必须：

- 安装 web 服务器
- 安装 PHP
- 安装数据库，比如 MySQL

官方的 PHP 网站 (PHP.net) 提供了 PHP 的安装说明：http://php.net/manual/zh/install.php

注释

提示：如需在 Windows 平台设置并立即运行 PHP，您还可以：

下载 WebMatrix

PHP 语法

PHP 脚本在服务器上执行，然后向浏览器发送回纯 **HTML** 结果。

基础 PHP 语法

PHP 脚本可放置于文档中的任何位置。

PHP 脚本以 *<?php* 开头，以 *?>* 结尾：

```
<?php
// 此处是 PHP 代码
?>
```

PHP 文件的默认文件扩展名是 ".php"。

PHP 文件通常包含 HTML 标签以及一些 PHP 脚本代码。

下面的例子是一个简单的 PHP 文件，其中包含了使用内建 PHP 函数 "echo" 在网页上输出文本 "Hello World!" 的一段 PHP 脚本：

实例

```
<!DOCTYPE html>
<html>
<body>

<h1>我的第一张 PHP 页面</h1>

<?php
echo "Hello World!";
?>

</body>
</html>
```

注释：PHP 语句以分号结尾（;）。PHP 代码块的关闭标签也会自动表明分号（因此在 PHP 代码块的最后一行不必使用分号）。

PHP 中的注释

PHP 代码中的注释不会被作为程序来读取和执行。它唯一的作用是供代码编辑者阅读。

注释用于：

- 使其他人理解您正在做的工作 - 注释可以让其他程序员了解您在每个步骤进行的工作（如果您供职于团队）
- 提醒自己做过什么 - 大多数程序员都曾经历过一两年后对项目进行返工，然后不得不重新考虑他们做过的事情。注释可以记录您在写代码时的思路。

PHP 支持三种注释：

实例

```
<!DOCTYPE html>
<html>
<body>

<?php
// 这是单行注释

# 这也是单行注释

/*
这是多行注释块
它横跨了
多行
*/
?>

</body>
</html>
```

PHP 大小写敏感

在 PHP 中，所有用户定义的函数、类和关键词（例如 if、else、echo 等等）都对大小写不敏感。

在下面的例子中，所有这三天 echo 语句都是合法的（等价）：

实例

```
<!DOCTYPE html>
<html>
<body>

<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>

</body>
</html>
```

不过在 PHP 中，所有变量都对大小写敏感。

在下面的例子中，只有第一条语句会显示 $color 变量的值（这是因为 $color、$COLOR 以及 $coLOR 被视作三个不同的变量）：

实例

```
<!DOCTYPE html>
<html>
<body>

<?php
$color="red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>

</body>
</html>
```

PHP 变量

变量是存储信息的容器：

实例

```
<?php
$x=5;
$y=6;
$z=$x+$y;
echo $z;
?>
```

类似代数

```
x=5
y=6
z=x+y
```

在代数中我们使用字母（比如 x）来保存值（比如 5）。

从上面的表达式 z=x+y，我们能够计算出 z 的值是 11。

在 PHP 中，这三个字母被称为变量。

注释：请把变量视为存储数据的容器。

PHP 变量

正如代数，PHP 变量可用于保存值（x=5）和表达式（z=x+y）。

变量的名称可以很短（比如 x 和 y），也可以取更具描述性的名称（比如 carname、total_volume）。

PHP 变量规则：

- 变量以 $ 符号开头，其后是变量的名称
- 变量名称必须以字母或下划线开头
- 变量名称不能以数字开头
- 变量名称只能包含字母数字字符和下划线（A-z、0-9 以及 _）
- 变量名称对大小写敏感（$y 与 $Y 是两个不同的变量）

注释：PHP 变量名称对大小写敏感！

创建 PHP 变量

PHP 没有创建变量的命令。

变量会在首次为其赋值时被创建：

实例

```
<?php
$txt="Hello world!";
$x=5;
$y=10.5;
?>
```

以上语句执行后，变量 txt 会保存值 Hello world!，变量 x 会保存值 5，变量 y 会保存值 10.5。

注释：如果您为变量赋的值是文本，请用引号包围该值。

PHP 是一门类型松散的语言

在上面的例子中，请注意我们不必告知 PHP 变量的数据类型。

PHP 根据它的值，自动把变量转换为正确的数据类型。

在诸如 C 和 C++ 以及 Java 之类的语言中，程序员必须在使用变量之前声明它的名称和类型。

PHP 变量作用域

在 PHP 中，可以在脚本的任意位置对变量进行声明。

变量的作用域指的是变量能够被引用/使用的那部分脚本。

PHP 有三种不同的变量作用域：

- local（局部）
- global（全局）
- static（静态）

Local 和 Global 作用域

函数之外声明的变量拥有 Global 作用域，只能在函数以外进行访问。

函数内部声明的变量拥有 LOCAL 作用域，只能在函数内部进行访问。

下面的例子测试了带有局部和全局作用域的变量：

实例

```
<?php
$x=5; // 全局作用域

function myTest() {
  $y=10; // 局部作用域
  echo "<p>测试函数内部的变量：</p>";
  echo "变量 x 是：$x";
  echo "<br>";
  echo "变量 y 是：$x";
}

myTest();

echo "<p>测试函数之外的变量：</p>";
echo "变量 x 是：$x";
echo "<br>";
echo "变量 y 是：$x";
?>
```

在上例中，有两个变量 $x 和 $y，以及一个函数 myTest()。$x 是全局变量，因为它是在函数之外声明的，而 $y 是局部变量，因为它是在函数内声明的。

如果我们在 myTest() 函数内部输出两个变量的值，$y 会输出在本地声明的值，但是无法 $x 的值，因为它在函数之外创建。

然后，如果在 myTest() 函数之外输出两个变量的值，那么会输出 $x 的值，但是不会输出 $y 的值，因为它是局部变量，并且在 myTest() 内部创建。

注释：您可以在不同的函数中创建名称相同的局部变量，因为局部变量只能被在其中创建它的函数识别。

PHP global 关键词

global 关键词用于访问函数内的全局变量。

要做到这一点，请在（函数内部）变量前面使用 global 关键词：

实例

```
<?php
$x=5;
$y=10;

function myTest() {
  global $x,$y;
  $y=$x+$y;
}

myTest();
echo $y; // 输出 15
?>
```

PHP 同时在名为 $GLOBALS[index] 的数组中存储了所有的全局变量。下标存有变量名。这个数组在函数内也可以访问，并能够用于直接更新全局变量。

上面的例子可以这样重写：

实例

```
<?php
$x=5;
$y=10;

function myTest() {
  $GLOBALS['y']=$GLOBALS['x']+$GLOBALS['y'];
}

myTest();
echo $y; // 输出 15
?>
```

PHP static 关键词

通常，当函数完成/执行后，会删除所有变量。不过，有时我需要不删除某个局部变量。实现这一点需要更进一步的工作。

要完成这一点，请在您首次声明变量时使用 *static* 关键词：

实例

```
<?php

function myTest() {
  static $x=0;
  echo $x;
  $x++;
}

myTest();
myTest();
myTest();

?>
```

然后，每当函数被调用时，这个变量所存储的信息都是函数最后一次被调用时所包含的信息。

注释：该变量仍然是函数的局部变量。

PHP 5 echo 和 print 语句

在 **PHP** 中，有两种基本的输出方法：**echo** 和 **print**。

在本教程中，我们几乎在每个例子中都会用到 echo 和 print。因此，本节为您讲解更多关于这两条输出语句的知识。

PHP echo 和 print 语句

echo 和 print 之间的差异：

- echo - 能够输出一个以上的字符串
- print - 只能输出一个字符串，并始终返回 1

提示：echo 比 print 稍快，因为它不返回任何值。

PHP echo 语句

echo 是一个语言结构，有无括号均可使用：echo 或 echo()。

显示字符串

下面的例子展示如何用 echo 命令来显示不同的字符串（同时请注意字符串中能包含 HTML 标记）：

```
<?php
echo "<h2>PHP is fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This", " string", " was", " made", " with multiple parameters
?>
```

显示变量

下面的例子展示如何用 echo 命令来显示字符串和变量：

```
<?php
$txt1="Learn PHP";
$txt2="W3School.com.cn";
$cars=array("Volvo","BMW","SAAB");

echo $txt1;
echo "<br>";
echo "Study PHP at $txt2";
echo "My car is a {$cars[0]}";
?>
```

PHP print 语句

print 也是语言结构，有无括号均可使用：print 或 print()。

显示字符串

下面的例子展示如何用 print 命令来显示不同的字符串（同时请注意字符串中能包含 HTML 标记）：

```
<?php
print "<h2>PHP is fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

显示变量

下面的例子展示如何用 print 命令来显示字符串和变量：

```
<?php
$txt1="Learn PHP";
$txt2="W3School.com.cn";
$cars=array("Volvo","BMW","SAAB");

print $txt1;
print "<br>";
print "Study PHP at $txt2";
print "My car is a {$cars[0]}";
?>
```

PHP 数据类型

字符串、整数、浮点数、逻辑、数组、对象、**NULL**。

PHP 字符串

字符串是字符序列，比如 "Hello world!"。

字符串可以是引号内的任何文本。您可以使用单引号或双引号：

实例

```
<?php
$x = "Hello world!";
echo $x;
echo "<br>";
$x = 'Hello world!';
echo $x;
?>
```

PHP 整数

整数是没有小数的数字。

整数规则：

- 整数必须有至少一个数字（0-9）
- 整数不能包含逗号或空格
- 整数不能有小数点
- 整数正负均可
- 可以用三种格式规定整数：十进制、十六进制（前缀是 0x）或八进制（前缀是 0）

在下面的例子中，我们将测试不同的数字。PHP var_dump() 会返回变量的数据类型和值：

实例

```
<?php
$x = 5985;
var_dump($x);
echo "<br>";
$x = -345; // 负数
var_dump($x);
echo "<br>";
$x = 0x8C; // 十六进制数
var_dump($x);
echo "<br>";
$x = 047; // 八进制数
var_dump($x);
?>
```

PHP 浮点数

浮点数是有小数点或指数形式的数字。

在下面的例子中，我们将测试不同的数字。PHP var_dump() 会返回变量的数据类型和值：

实例

```
<?php
$x = 10.365;
var_dump($x);
echo "<br>";
$x = 2.4e3;
var_dump($x);
echo "<br>";
$x = 8E-5;
var_dump($x);
?>
```

PHP 逻辑

逻辑是 true 或 false。

```
$x=true;
$y=false;
```

逻辑常用于条件测试。您将在本教程稍后的章节学到更多有关条件测试的知识。

PHP 数组

数组在一个变量中存储多个值。

在下面的例子中，我们将测试不同的数组。PHP var_dump() 会返回变量的数据类型和值：

实例

```
<?php
$cars=array("Volvo","BMW","SAAB");
var_dump($cars);
?>
```

您将在本教程稍后的章节学到更多有关数组的知识。

PHP 对象

对象是存储数据和有关如何处理数据的信息的数据类型。

在 PHP 中，必须明确地声明对象。

首先我们必须声明对象的类。对此，我们使用 class 关键词。类是包含属性和方法的结构。

然后我们在对象类中定义数据类型，然后在该类的实例中使用此数据类型：

实例

```
<?php
class Car
{
  var $color;
  function Car($color="green") {
    $this->color = $color;
  }
  function what_color() {
    return $this->color;
  }
}
?>
```

您将在本教程稍后的章节学到更多有关对象的知识。

PHP NULL 值

特殊的 NULL 值表示变量无值。NULL 是数据类型 NULL 唯一可能的值。

NULL 值标示变量是否为空。也用于区分空字符串与空值数据库。

可以通过把值设置为 NULL，将变量清空：

实例

```
<?php
$x="Hello world!";
$x=null;
var_dump($x);
?>
```

PHP 字符串函数

字符串是字符序列，比如 **"Hello world!"**。

PHP 字符串函数

在本节中，我们将学习常用的字符串操作函数。

PHP strlen() 函数

strlen() 函数返回字符串的长度，以字符计。

下例返回字符串 "Hello world!" 的长度：

实例

```
<?php
echo strlen("Hello world!");
?>
```

以上代码的输出是：12

提示：strlen() 常用于循环和其他函数，在确定字符串何时结束很重要时。（例如，在循环中，我们也许需要在字符串的最后一个字符之后停止循环）。

PHP strpos() 函数

strpos() 函数用于检索字符串内指定的字符或文本。

如果找到匹配，则会返回首个匹配的字符位置。如果未找到匹配，则将返回 FALSE。

下例检索字符串 "Hello world!" 中的文本 "world"：

实例

```
<?php
echo strpos("Hello world!","world");
?>
```

以上代码的输出是：6。

提示：上例中字符串 "world" 的位置是 6。是 6（而不是 7）的理由是，字符串中首字符的位置是 0 而不是 1。

完整的 PHP String 参考手册

如需完整的字符串函数参考手册，请访问我们的 PHP String 参考手册。

这个手册提供了每个函数的简要描述和实例！

PHP 常量

常量类似变量，但是常量一旦被定义就无法更改或撤销定义。

PHP 常量

常量是单个值的标识符（名称）。在脚本中无法改变该值。

有效的常量名以字符或下划线开头（常量名称前面没有 $ 符号）。

注释：与变量不同，常量贯穿整个脚本是自动全局的。

设置 PHP 常量

如需设置常量，请使用 define() 函数 - 它使用三个参数：

1. 首个参数定义常量的名称
2. 第二个参数定义常量的值
3. 可选的第三个参数规定常量名是否对大小写敏感。默认是 false。

下例创建了一个对大小写敏感的常量，值为 "Welcome to W3School.com.cn!"：

实例

```
<?php
define("GREETING", "Welcome to W3School.com.cn!");
echo GREETING;
?>
```

下例创建了一个对大小写不敏感的常量，值为 "Welcome to W3School.com.cn!"：

实例

```
<?php
define("GREETING", "Welcome to W3School.com.cn!", true);
echo greeting;
?>
```

PHP 运算符

本节展示了可用于 **PHP** 脚本中的各种运算符.

PHP 算数运算符

运算符	名称	例子	结果
+	加法	$x + $y	$x 与 $y 求和
-	减法	$x - $y	$x 与 $y 的差数
*	乘法	$x * $y	$x 与 $y 的乘积
/	除法	$x / $y	$x 与 $y 的商数
%	模数	$x % $y	$x 除 $y 的余数

下例展示了使用不同算数运算符的不同结果：

实例

```
<?php
$x=10;
$y=6;
echo ($x + $y); // 输出 16
echo ($x - $y); // 输出 4
echo ($x * $y); // 输出 60
echo ($x / $y); // 输出 1.6666666666667
echo ($x % $y); // 输出 4
?>
```

PHP 赋值运算符

PHP 赋值运算符用于向变量写值。

PHP 中基础的赋值运算符是 "="。这意味着右侧赋值表达式会为左侧运算数设置值。

赋值	等同于	描述
x = y	x = y	右侧表达式为左侧运算数设置值。
x += y	x = x + y	加
x -= y	x = x - y	减
x *= y	x = x * y	乘
x /= y	x = x / y	除
x %= y	x = x % y	模数

下例展示了使用不同赋值运算符的不同结果：

实例

```
<?php
$x=10;
echo $x; // 输出 10

$y=20;
$y += 100;
echo $y; // 输出 120

$z=50;
$z -= 25;
echo $z; // 输出 25

$i=5;
$i *= 6;
echo $i; // 输出 30

$j=10;
$j /= 5;
echo $j; // 输出 2

$k=15;
$k %= 4;
echo $k; // 输出 3
?>
```

PHP 字符串运算符

运算符	名称	例子	结果
.	串接	$txt1 = "Hello" $txt2 = $txt1 . " world!"	现在 $txt2 包含 "Hello world!"
.=	串接赋值	$txt1 = "Hello" $txt1 .= " world!"	现在 $txt1 包含 "Hello world!"

下例展示了使用字符串运算符的结果：

实例

```
<?php
$a = "Hello";
$b = $a . " world!";
echo $b; // 输出 Hello world!

$x="Hello";
$x .= " world!";
echo $x; // 输出 Hello world!
?>
```

PHP 递增/递减运算符

运算符	名称	描述
++$x	前递增	$x 加一递增，然后返回 $x
$x++	后递增	返回 $x，然后 $x 加一递增
--$x	前递减	$x 减一递减，然后返回 $x
$x--	后递减	返回 $x，然后 $x 减一递减

下例展示了使用不同递增/递减运算符的不同结果：

实例

```
<?php
$x=10;
echo ++$x; // 输出 11

$y=10;
echo $y++; // 输出 10

$z=5;
echo --$z; // 输出 4

$i=5;
echo $i--; // 输出 5
?>
```

PHP 比较运算符

PHP 比较运算符用于比较两个值（数字或字符串）：

运算符	名称	例子	结果
==	等于	$x == $y	如果 $x 等于 $y，则返回 true。
===	全等（完全相同）	$x === $y	如果 $x 等于 $y，且它们类型相同，则返回 true。
!=	不等于	$x != $y	如果 $x 不等于 $y，则返回 true。
<>	不等于	$x <> $y	如果 $x 不等于 $y，则返回 true。
!==	不全等（完全不同）	$x !== $y	如果 $x 不等于 $y，且它们类型不相同，则返回 true。
>	大于	$x > $y	如果 $x 大于 $y，则返回 true。
<	大于	$x < $y	如果 $x 小于 $y，则返回 true。
>=	大于或等于	$x >= $y	如果 $x 大于或者等于 $y，则返回 true.
<=	小于或等于	$x <= $y	如果 $x 小于或者等于 $y，则返回 true。

下例展示了使用某些比较运算符的不同结果：

实例

```
<?php
$x=100;
$y="100";

var_dump($x == $y);
echo "<br>";
var_dump($x === $y);
echo "<br>";
var_dump($x != $y);
echo "<br>";
var_dump($x !== $y);
echo "<br>";

$a=50;
$b=90;

var_dump($a > $b);
echo "<br>";
var_dump($a < $b);
?>
```

PHP 逻辑运算符

运算符	名称	例子	结果
and	与	$x and $y	如果 $x 和 $y 都为 true，则返回 true。
or	或	$x or $y	如果 $x 和 $y 至少有一个为 true，则返回 true。
xor	异或	$x xor $y	如果 $x 和 $y 有且仅有一个为 true，则返回 true。
&&	与	$x && $y	如果 $x 和 $y 都为 true，则返回 true。
||	或	$x || $y	如果 $x 和 $y 至少有一个为 true，则返回 true。
!	非	!$x	如果 $x 不为 true，则返回 true。

PHP 数组运算符

PHP 数组运算符用于比较数组：

运算符	名称	例子	结果
+	联合	$x + $y	$x 和 $y 的联合（但不覆盖重复的键）
==	相等	$x == $y	如果 $x 和 $y 拥有相同的键/值对，则返回 true。
===	全等	$x === $y	如果 $x 和 $y 拥有相同的键/值对，且顺序相同类型相同，则返回 true。
!=	不相等	$x != $y	如果 $x 不等于 $y，则返回 true。
<>	不相等	$x <> $y	如果 $x 不等于 $y，则返回 true。
!==	不全等	$x !== $y	如果 $x 与 $y 完全不同，则返回 true。

下例展示了使用不同数组运算符的不同结果：

实例

```
<?php
$x = array("a" => "red", "b" => "green");
$y = array("c" => "blue", "d" => "yellow");
$z = $x + $y; // $x 与 $y 的联合
var_dump($z);
var_dump($x == $y);
var_dump($x === $y);
var_dump($x != $y);
var_dump($x <> $y);
var_dump($x !== $y);
?>
```

PHP if...else...elseif 语句

条件语句用于基于不同条件执行不同的动作

PHP 条件语句

在您编写代码时，经常会希望为不同的决定执行不同的动作。您可以在代码中使用条件语句来实现这一点。

在 PHP 中，我们可以使用以下条件语句：

- *if* 语句 - 如果指定条件为真，则执行代码
- *if...else* 语句 - 如果条件为 true，则执行代码；如果条件为 false，则执行另一端代码
- *if...elseif....else* 语句 - 选择若干段代码块之一来执行
- *switch* 语句 - 语句多个代码块之一来执行

PHP - if 语句

if 语句用于在指定条件为 *true* 时执行代码。

语法

```
if (条件) {
  当条件为 true 时执行的代码;
}
```

下例将输出 "Have a good day!"，如果当前时间 (HOUR) 小于 20：

实例

```
<?php
$t=date("H");

if ($t<"20") {
  echo "Have a good day!";
}
?>
```

PHP - if...else 语句

请使用 if....else 语句在条件为 *true* 时执行代码，在条件为 *false* 时执行另一段代码。

语法

```
if (条件) {
  条件为 true 时执行的代码;
} else {
  条件为 false 时执行的代码;
}
```

下例将输出 "Have a good day!"，如果当前时间 (HOUR) 小于 20，否则输出 "Have a good night!"：

实例

```
<?php
$t=date("H");

if ($t<"20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>
```

PHP - if...elseif....else 语句

请使用 if....elseif...else 语句来选择若干代码块之一来执行。

语法

```
if (条件) {
  条件为 true 时执行的代码;
} elseif (condition) {
  条件为 true 时执行的代码;
} else {
  条件为 false 时执行的代码;
}
```

下例将输出 "Have a good morning!"，如果当前时间 (HOUR) 小于 10，如果当前时间小于 20，则输出 "Have a good day!"。否则将输出 "Have a good night!"：

实例

```
<?php
$t=date("H");

if ($t<"10") {
  echo "Have a good morning!";
} elseif ($t<"20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>
```

PHP - switch 语句

我们在下一节中学习 switch 语句。

PHP Switch 语句

switch 语句用于基于不同条件执行不同动作。

Switch 语句

如果您希望有选择地执行若干代码块之一，请使用 Switch 语句。

使用 Switch 语句可以避免冗长的 if..elseif..else 代码块。

语法

```
switch (expression)
{
case label1:
  code to be executed if expression = label1;
  break;
case label2:
  code to be executed if expression = label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

工作原理：

1. 对表达式（通常是变量）进行一次计算
2. 把表达式的值与结构中 case 的值进行比较
3. 如果存在匹配，则执行与 case 关联的代码
4. 代码执行后，*break* 语句阻止代码跳入下一个 case 中继续执行
5. 如果没有 case 为真，则使用 default 语句

实例

```
<?php
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>

</body>
</html>
```

PHP while 循环

PHP while 循环在指定条件为 **true** 时执行代码块。

PHP 循环

在您编写代码时，经常需要反复运行同一代码块。我们可以使用循环来执行这样的任务，而不是在脚本中添加若干几乎相等的代码行。

在 PHP 中，我们有以下循环语句：

- *while* - 只要指定条件为真，则循环代码块
- *do...while* - 先执行一次代码块，然后只要指定条件为真则重复循环
- *for* - 循环代码块指定次数
- *foreach* - 遍历数组中的每个元素并循环代码块

PHP while 循环

只要指定的条件为真，while 循环就会执行代码块。

语法

```
while (条件为真) {
  要执行的代码;
}
```

上例首先把变量 $x 设置为 1（$x=1）。然后执行 while 循环，只要 $x 小于或等于 5。循环每运行一次，$x 将递增 1：

实例

```
<?php
$x=1;

while($x<=5) {
  echo "这个数字是：$x <br>";
  $x++;
}
?>
```

PHP do...while 循环

do...while 循环首先会执行一次代码块，然后检查条件，如果指定条件为真，则重复循环。

语法

```
do {
  要执行的代码;
} while (条件为真);
```

下面的例子首先把变量 $x 设置为 1（$x=1）。然后，do while 循环输出一段字符串，然后对变量 $x 递增 1。随后对条件进行检查（$x 是否小于或等于 5）。只要 $x 小于或等于 5，循环将会继续运行：

实例

```
<?php
$x=1;

do {
  echo "这个数字是：$x <br>";
  $x++;
} while ($x<=5);
?>
```

请注意，do while 循环只在执行循环内的语句之后才对条件进行测试。这意味着 do while 循环至少会执行一次语句，即使条件测试在第一次就失败了。

下面的例子把 $x 设置为 6，然后运行循环，随后对条件进行检查：

实例

```
<?php
$x=6;

do {
  echo "这个数字是：$x <br>";
  $x++;
} while ($x<=5);
?>
```

下一节会讲解 for 循环和 foreach 循环。

PHP for 循环

PHP for 循环执行代码块指定的次数。

PHP for 循环

如果您已经提前确定脚本运行的次数，可以使用 for 循环。

语法

```
for (init counter; test counter; increment counter) {
  code to be executed;
}
```

参数：

- init counter：初始化循环计数器的值
- test counter：: 评估每个循环迭代。如果值为 TRUE，继续循环。如果它的值为 FALSE，循环结束。
- increment counter：增加循环计数器的值

下面的例子显示了从 0 到 10 的数字：

实例

```
<?php
for ($x=0; $x<=10; $x++) {
  echo "数字是：$x <br>";
}
?>
```

PHP foreach 循环

foreach 循环只适用于数组，并用于遍历数组中的每个键/值对。

语法

```
foreach ($array as $value) {
  code to be executed;
}
```

每进行一次循环迭代，当前数组元素的值就会被赋值给 $value 变量，并且数组指针会逐一地移动，直到到达最后一个数组元素。

下面的例子演示的循环将输出给定数组（$colors）的值：

实例

```
<?php
$colors = array("red","green","blue","yellow");

foreach ($colors as $value) {
  echo "$value <br>";
}
?>
```

在稍后的章节，您将学到更多有关数组的知识。

PHP 函数

PHP 的真正力量来自它的函数：它拥有超过 **1000** 个内建的函数。

PHP 用户定义函数

除了内建的 PHP 函数，我们可以创建我们自己的函数。

函数是可以在程序中重复使用的语句块。

页面加载时函数不会立即执行。

函数只有在被调用时才会执行。

在 PHP 创建用户定义函数

用户定义的函数声明以关单 "function" 开头：

语法

```
function functionName() {
  被执行的代码;
}
```

注释：函数名能够以字母或下划线开头（而非数字）。

注释：函数名对大小写不敏感。

提示：函数名应该能够反映函数所执行的任务。

在下面的例子中，我们创建名为 "writeMsg()" 的函数。打开的花括号（{）指示函数代码的开始，而关闭的花括号（}）指示函数的结束。此函数输出 "Hello world!"。如需调用该函数，只要使用函数名即可：

实例

```
<?php
function writeMsg() {
  echo "Hello world!";
}

writeMsg(); // 调用函数
?>
```

PHP 函数参数

可以通过参数向函数传递信息。参数类似变量。

参数被定义在函数名之后，括号内部。您可以添加任意多参数，只要用逗号隔开即可。

下面的例子中的函数有一个参数（$fname）。当调用 familyName() 函数时，我们同时要传递一个名字（例如 Bill），这样会输出不同的名字，但是姓氏相同：

实例

```
<?php
function familyName($fname) {
  echo "$fname Zhang.<br>";
}

familyName("Li");
familyName("Hong");
familyName("Tao");
familyName("Xiao Mei");
familyName("Jian");
?>
```

下面的例子中的函数有两个参数（$fname 和 $year）：

实例

```
<?php
function familyName($fname,$year) {
  echo "$fname Zhang. Born in $year <br>";
}

familyName("Li","1975");
familyName("Hong","1978");
familyName("Tao","1983");
?>
```

PHP 默认参数值

下面的例子展示了如何使用默认参数。如果我们调用没有参数的 setHeight() 函数，它的参数会取默认值：

实例

```
<?php
function setHeight($minheight=50) {
  echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // 将使用默认值 50
setHeight(135);
setHeight(80);
?>
```

PHP 函数 - 返回值

如需使函数返回值，请使用 return 语句：

实例

```
<?php
function sum($x,$y) {
  $z=$x+$y;
  return $z;
}

echo "5 + 10 = " . sum(5,10) . "<br>";
echo "7 + 13 = " . sum(7,13) . "<br>";
echo "2 + 4 = " . sum(2,4);
?>
```

PHP 数组

数组能够在单独的变量名中存储一个或多个值。

实例

数组在单个变量中存储多个值：

```
<?php
$cars=array("Volvo","BMW","SAAB");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] .
?>
```

什么是数组？

数组是特殊的变量，它可以同时保存一个以上的值。

如果您有一个项目列表（例如汽车品牌列表），在单个变量中存储这些品牌名称是这样的：

```
$cars1="Volvo";
$cars2="BMW";
$cars3="SAAB";
```

不过，假如您希望对变量进行遍历并找出特定的那个值？或者如果您需要存储 300 个汽车品牌，而不是 3 个呢？

解决方法是创建数组！

数组能够在单一变量名中存储许多值，并且您能够通过引用下标号来访问某个值。

在 PHP 中创建数组

在 PHP 中， array() 函数用于创建数组：

```
array();
```

在 PHP 中，有三种数组类型：

- 索引数组 - 带有数字索引的数组

- 关联数组 - 带有指定键的数组
- 多维数组 - 包含一个或多个数组的数组

PHP 索引数组

有两种创建索引数组的方法：

索引是自动分配的（索引从 0 开始）：

```
$cars=array("Volvo","BMW","SAAB");
```

或者也可以手动分配索引：

```
$cars[0]="Volvo";
$cars[1]="BMW";
$cars[2]="SAAB";
```

下面的例子创建名为 $cars 的索引数组，为其分配三个元素，然后输出包含数组值的一段文本：

实例

```
<?php
$cars=array("Volvo","BMW","SAAB");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] .
?>
```

获得数组的长度 - count() 函数

count() 函数用于返回数组的长度（元素数）：

实例

```
<?php
$cars=array("Volvo","BMW","SAAB");
echo count($cars);
?>
```

遍历索引数组

如需遍历并输出索引数组的所有值，您可以使用 for 循环，就像这样：

实例

```
<?php
$cars=array("Volvo","BMW","SAAB");
$arrlength=count($cars);

for($x=0;$x<$arrlength;$x++) {
  echo $cars[$x];
  echo "<br>";
}
?>
```

PHP 关联数组

关联数组是使用您分配给数组的指定键的数组。

有两种创建关联数组的方法：

```
$age=array("Peter"=>"35","Ben"=>"37","Joe"=>"43");
```

或者：

```
$age['Peter']="35";
$age['Ben']="37";
$age['Joe']="43";
```

随后可以在脚本中使用指定键：

实例

```
<?php
$age=array("Bill"=>"35","Steve"=>"37","Peter"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

遍历关联数组

如需遍历并输出关联数组的所有值，您可以使用 foreach 循环，就像这样：

实例

```
<?php
$age=array("Bill"=>"35","Steve"=>"37","Peter"=>"43");

foreach($age as $x=>$x_value) {
  echo "Key=" . $x . ", Value=" . $x_value;
  echo "<br>";
}
?>
```

多维数组

我们将在 PHP 高级教程中讲解多维数组。

完整的 PHP 数组参考手册

如需完整的数组函数参考手册，请访问我们的 PHP 数组参考手册。

该参考手册包含每个函数的简要描述、使用示例。

PHP 数组排序

数组中的元素能够以字母或数字顺序进行升序或降序排序。

PHP - 数组的排序函数

在本节中，我们将学习如下 PHP 数组排序函数：

- sort() - 以升序对数组排序
- rsort() - 以降序对数组排序
- asort() - 根据值，以升序对关联数组进行排序
- ksort() - 根据键，以升序对关联数组进行排序
- arsort() - 根据值，以降序对关联数组进行排序
- krsort() - 根据键，以降序对关联数组进行排序

对数组进行升序排序 - sort()

下面的例子按照字母升序对数组 $cars 中的元素进行排序：

实例

```
<?php
$cars=array("Volvo","BMW","SAAB");
sort($cars);
?>
```

下面的例子按照数字升序对数组 $numbers 中的元素进行排序：

实例

```
<?php
$numbers=array(3,5,1,22,11);
sort($numbers);
?>
```

对数组进行降序排序 - rsort()

下面的例子按照字母降序对数组 $cars 中的元素进行排序：

实例

```
<?php
$cars=array("Volvo","BMW","SAAB");
rsort($cars);
?>
```

下面的例子按照数字降序对数组 $numbers 中的元素进行排序：

实例

```
<?php
$numbers=array(3,5,1,22,11);
rsort($numbers);
?>
```

根据值对数组进行升序排序 - asort()

下面的例子根据值对关联数组进行升序排序：

实例

```
<?php
$age=array("Bill"=>"35","Steve"=>"37","Peter"=>"43");
asort($age);
?>
```

根据键对数组进行升序排序 - ksort()

下面的例子根据键对关联数组进行升序排序：

实例

```
<?php
$age=array("Bill"=>"35","Steve"=>"37","Peter"=>"43");
ksort($age);
?>
```

根据值对数组进行降序排序 - arsort()

下面的例子根据值对关联数组进行降序排序：

实例

```
<?php
$age=array("Bill"=>"35","Steve"=>"37","Peter"=>"43");
arsort($age);
?>
```

根据键对数组进行降序排序 - krsort()

下面的例子根据键对关联数组进行降序排序：

实例

```
<?php
$age=array("Bill"=>"35","Steve"=>"37","Peter"=>"43");
krsort($age);
?>
```

完整的 PHP 数组参考手册

如需完整的数组函数参考手册，请访问我们的 PHP 数组参考手册。

该参考手册包含每个函数的简要描述、使用示例。

PHP 全局变量 - 超全局变量

")

超全局变量 在 **PHP 4.1.0** 中引入，是在全部作用域中始终可用的内置变量。

PHP 全局变量 - 超全局变量

PHP 中的许多预定义变量都是“超全局的”，这意味着它们在一个脚本的全部作用域中都可用。在函数或方法中无需执行 global $variable; 就可以访问它们。

这些超全局变量是：

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE
- $_SESSION

本节会介绍一些超全局变量，并会在稍后的章节讲解其他的超全局变量。

$GLOBALS — 引用全局作用域中可用的全部变量

$GLOBALS 这种全局变量用于在 PHP 脚本中的任意位置访问全局变量（从函数或方法中均可）。

PHP 在名为 $GLOBALS[index] 的数组中存储了所有全局变量。变量的名字就是数组的键。

下面的例子展示了如何使用超级全局变量 $GLOBALS：

实例

```
<?php
$x = 75;
$y = 25;

function addition() {
  $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>
```

在上面的例子中，由于 z 是 $GLOBALS 数组中的变量，因此在函数之外也可以访问它。

PHP $_SERVER

$_SERVER 这种超全局变量保存关于报头、路径和脚本位置的信息。

下面的例子展示了如何使用 $_SERVER 中的某些元素：

实例

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

下表列出了您能够在 $_SERVER 中访问的最重要的元素：

元素/代码	描述
$_SERVER['PHP_SELF']	返回当前执行脚本的文件名。
$_SERVER['GATEWAY_INTERFACE']	返回服务器使用的 CGI 规范的版本。
	返回当前运行脚本所在的服务器

	的 IP 地址。
$_SERVER['SERVER_NAME']	返回当前运行脚本所在的服务器的主机名（比如 www.w3school.com.cn）。
$_SERVER['SERVER_SOFTWARE']	返回服务器标识字符串（比如 Apache/2.2.24）。
$_SERVER['SERVER_PROTOCOL']	返回请求页面时通信协议的名称和版本（例如，“HTTP/1.0”）。
$_SERVER['REQUEST_METHOD']	返回访问页面使用的请求方法（例如 POST）。
$_SERVER['REQUEST_TIME']	返回请求开始时的时间戳（例如 1577687494）。
$_SERVER['QUERY_STRING']	返回查询字符串，如果是通过查询字符串访问此页面。
$_SERVER['HTTP_ACCEPT']	返回来自当前请求的请求头。
$_SERVER['HTTP_ACCEPT_CHARSET']	返回来自当前请求的 Accept_Charset 头（例如 utf-8,ISO-8859-1）
$_SERVER['HTTP_HOST']	返回来自当前请求的 Host 头。
$_SERVER['HTTP_REFERER']	返回当前页面的完整 URL（不可靠，因为不是所有用户代理都支持）。
$_SERVER['HTTPS']	是否通过安全 HTTP 协议查询脚本。
$_SERVER['REMOTE_ADDR']	返回浏览当前页面的用户的 IP 地址。
$_SERVER['REMOTE_HOST']	返回浏览当前页面的用户的主机名。
$_SERVER['REMOTE_PORT']	返回用户机器上连接到 Web 服务器所使用的端口号。
$_SERVER['SCRIPT_FILENAME']	返回当前执行脚本的绝对路径。
$_SERVER['SERVER_ADMIN']	该值指明了 Apache 服务器配置文件中的 SERVER_ADMIN 参数。
$_SERVER['SERVER_PORT']	Web 服务器使用的端口。默认值为 “80”。
$_SERVER['SERVER_SIGNATURE']	返回服务器版本和虚拟主机名。

$_SERVER['SERVER_SIGNATURE']	返回服务器版本和虚拟主机名。
$_SERVER['PATH_TRANSLATED']	当前脚本所在文件系统（非文档根目录）的基本路径。
$_SERVER['SCRIPT_NAME']	返回当前脚本的路径。
$_SERVER['SCRIPT_URI']	返回当前页面的 URI。

PHP $_REQUEST

PHP $_REQUEST 用于收集 HTML 表单提交的数据。

下面的例子展示了一个包含输入字段及提交按钮的表单。当用户通过点击提交按钮来提交表单数据时, 表单数据将发送到 <form> 标签的 action 属性中指定的脚本文件。在这个例子中，我们指定文件本身来处理表单数据。如果您需要使用其他的 PHP 文件来处理表单数据，请修改为您选择的文件名即可。然后，我们可以使用超级全局变量 $_REQUEST 来收集 input 字段的值：

实例

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>

<?php
$name = $_REQUEST['fname'];
echo $name;
?>

</body>
</html>
```

PHP $_POST

PHP $_POST 广泛用于收集提交 method="post" 的 HTML 表单后的表单数据。$_POST 也常用于传递变量。

下面的例子展示了一个包含输入字段和提交按钮的表单。当用户点击提交按钮来提交数据后，表单数据会发送到 <form> 标签的 action 属性中指定的文件。在本例中，我们指定文件本身来处理表单数据。如果您希望使用另一个 PHP 页面来处理表单数据，请用更改为您选择的文件名。然后，我们可以使用超全局变量 $_POST 来收集输入字段的值：

实例

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
Name: <input type="text" name="fname">
<input type="submit">
</form>

<?php
$name = $_POST['fname'];
echo $name;
?>

</body>
</html>
```

PHP $_GET

PHP $_GET 也可用于收集提交 HTML 表单 (method="get") 之后的表单数据。

$_GET 也可以收集 URL 中的发送的数据。

假设我们有一张页面含有带参数的超链接：

```
<html>
<body>

<a href="test_get.php?subject=PHP&web=W3school.com.cn">测试 $GET</a

</body>
</html>
```

当用户点击链接 "Test $GET"，参数 "subject" 和 "web" 被发送到 "test_get.php"，然后您就能够通过 $_GET 在 "test_get.php" 中访问这些值了。

下面的例子是 "test_get.php" 中的代码：

实例

```
<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

</body>
</html>
```

提示：您将在 PHP 表单这一节中学到更多有关 $_POST 和 $_GET 的知识。

")

PHP 魔术变量

PHP 向它运行的任何脚本提供了大量的预定义常量。

不过很多常量都是由不同的扩展库定义的，只有在加载了这些扩展库时才会出现，或者动态加载后，或者在编译时已经包括进去了。

有八个魔术常量它们的值随着它们在代码中的位置改变而改变。

例如 __LINE__ 的值就依赖于它在脚本中所处的行来决定。这些特殊的常量不区分大小写，如下：

__LINE__

文件中的当前行号。

实例:

```
<?php
echo '这是第 “ '  . __LINE__ . ' ” 行';
?>
```

以上实例输出结果为：

```
这是第 “ 2 ” 行
```

__FILE__

文件的完整路径和文件名。如果用在被包含文件中，则返回被包含的文件名。

自 PHP 4.0.2 起，**FILE** 总是包含一个绝对路径（如果是符号连接，则是解析后的绝对路径），而在此之前的版本有时会包含一个相对路径。

实例:

```
<?php
echo '该文件位于 “ '  . __FILE__ . ' ” ';
?>
```

以上实例输出结果为：

```
该文件位于 “ E:\wamp\www\test\index.php ”
```

__DIR__

文件所在的目录。如果用在被包括文件中，则返回被包括的文件所在的目录。

它等价于 dirname(**FILE**)。除非是根目录，否则目录中名不包括末尾的斜杠。（PHP 5.3.0中新增）

实例:

```
<?php
echo '该文件位于 “ '  . __DIR__ . ' ” ';
?>
```

以上实例输出结果为：

```
该文件位于 “ E:\wamp\www\test ”
```

__FUNCTION__

函数名称（PHP 4.3.0 新加）。自 PHP 5 起本常量返回该函数被定义时的名字（区分大小写）。在 PHP 4 中该值总是小写字母的。

实例:

```
<?php
function test() {
    echo  '函数名为：' . __FUNCTION__ ;
}
test();
?>
```

以上实例输出结果为：

```
函数名为：test
```

__CLASS__

类的名称（PHP 4.3.0 新加）。自 PHP 5 起本常量返回该类被定义时的名字（区分大小写）。

在 PHP 4 中该值总是小写字母的。类名包括其被声明的作用区域（例如 Foo\Bar）。注意自 PHP 5.4 起 **CLASS** 对 trait 也起作用。当用在 trait 方法中时，**CLASS** 是调用 trait 方法的类的名字。

实例:

```
<?php
<?php
class test {
    function _print() {
        echo '类名为：'  . __CLASS__ . "<br>";
        echo  '函数名为：' . __FUNCTION__ ;
    }
}
$t = new test();
$t->_print();
?>
```

以上实例输出结果为：

```
类名为：test
函数名为：_print
```

__TRAIT__

Trait 的名字（PHP 5.4.0 新加）。自 PHP 5.4.0 起，PHP 实现了代码复用的一个方法，称为 traits。

Trait 名包括其被声明的作用区域（例如 Foo\Bar）。

从基类继承的成员被插入的 SayWorld Trait 中的 MyHelloWorld 方法所覆盖。其行为 MyHelloWorld 类中定义的方法一致。优先顺序是当前类中的方法会覆盖 trait 方法，而 trait 方法又覆盖了基类中的方法。

```
<?php
class Base {
    public function sayHello() {
        echo 'Hello ';
    }
}

trait SayWorld {
    public function sayHello() {
        parent::sayHello();
        echo 'World!';
    }
}

class MyHelloWorld extends Base {
    use SayWorld;
}

$o = new MyHelloWorld();
$o->sayHello();
?>
```

以上例程会输出：

```
Hello World!
```

__METHOD__

类的方法名（PHP 5.0.0 新加）。返回该方法被定义时的名字（区分大小写）。

实例:

```
<?php
function test() {
    echo  '函数名为：' . __METHOD__ ;
}
test();
?>
```

以上实例输出结果为：

```
函数名为：test
```

__NAMESPACE__

当前命名空间的名称（区分大小写）。此常量是在编译时定义的（PHP 5.3.0 新增）。

实例:

```
<?php
namespace MyProject;

echo '命名空间为："', __NAMESPACE__, '"'; // 输出 "MyProject"
?>
```

以上实例输出结果为：

```
命名空间为："MyProject"
```

PHP 命名空间(namespace)

PHP 命名空间(namespace)是在PHP 5.3中加入的，如果你学过C#和Java，那命名空间就不算什么新事物。 不过在PHP当中还是有着相当重要的意义。

PHP 命名空间可以解决以下两类问题：

1. 用户编写的代码与PHP内部的类/函数/常量或第三方类/函数/常量之间的名字冲突。
2. 为很长的标识符名称(通常是为了缓解第一类问题而定义的)创建一个别名（或简短）的名称，提高源代码的可读性。

定义命名空间

默认情况下，所有常量、类和函数名都放在全局空间下，就和PHP支持命名空间之前一样。

命名空间通过关键字namespace 来声明。如果一个文件中包含命名空间，它必须在其它所有代码之前声明命名空间。语法格式如下；

```
< ?php
// 定义代码在 'MyProject' 命名空间中
namespace MyProject;

// ... 代码 ...
```

你也可以在同一个文件中定义不同的命名空间代码，如：

```
< ?php
namespace MyProject1;
// MyProject1 命名空间中的PHP代码

namespace MyProject2;
// MyProject2 命名空间中的PHP代码

// 另一种语法
namespace MyProject3 {
 // MyProject3 命名空间中的PHP代码
}
?>
```

在声明命名空间之前唯一合法的代码是用于定义源文件编码方式的 declare 语句。所有非 PHP 代码包括空白符都不能出现在命名空间的声明之前。

```
<?php
declare(encoding='UTF-8'); //定义多个命名空间和不包含在命名空间中的代码
namespace MyProject {

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
}

namespace { // 全局代码
session_start();
$a = MyProject\connect();
echo MyProject\Connection::start();
}
?>
```

以下代码会出现语法错误：

```
<html>
<?php
namespace MyProject; // 命名空间前出现了“<html>” 会致命错误 -  命名空间必
?>
```

子命名空间

与目录和文件的关系很象，PHP 命名空间也允许指定层次化的命名空间的名称。因此，命名空间的名字可以使用分层次的方式定义：

```
<?php
namespace MyProject\Sub\Level;  //声明分层次的单个命名空间

const CONNECT_OK = 1;
class Connection { /* ... */ }
function Connect() { /* ... */  }

?>
```

上面的例子创建了常量 MyProject\Sub\Level\CONNECT_OK，类 MyProject\Sub\Level\Connection 和函数 MyProject\Sub\Level\Connection。

命名空间使用

PHP 命名空间中的类名可以通过三种方式引用：

1. 非限定名称，或不包含前缀的类名称，例如 $a=new foo(); 或 foo::staticmethod();。如果当前命名空间是 currentnamespace，foo 将被解析为 currentnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，则 foo 会被解析为foo。 警告：如果命名空间中的函数或常量未定义，则该非限定的函数名称或常量名称会被解析为全局函数名称或常量名称。

2. 限定名称,或包含前缀的名称，例如 $a = new subnamespace\foo(); 或 subnamespace\foo::staticmethod();。如果当前的命名空间是 currentnamespace，则 foo 会被解析为 currentnamespace\subnamespace\foo。如果使用 foo 的代码是全局的，不包含在任何命名空间中的代码，foo 会被解析为subnamespace\foo。 完全限定名称，或包含了全局前缀操作符的名称，例如， $a = new \currentnamespace\foo(); 或 \currentnamespace\foo::staticmethod();。在这种情况下，foo 总是被解析为代码中的文字名(literal name)currentnamespace\foo。

下面是一个使用这三种方式的实例：

file1.php 文件代码

```
<?php
namespace Foo\Bar\subnamespace;

const FOO = 1;
function foo() {}
class foo
{
    static function staticmethod() {}
}
?>
```

file2.php 文件代码

```
<?php
namespace Foo\Bar;
include 'file1.php';

const FOO = 2;
function foo() {}
class foo
{
    static function staticmethod() {}
}

/* 非限定名称 */
foo(); // 解析为 Foo\Bar\foo resolves to function Foo\Bar\foo
foo::staticmethod(); // 解析为类 Foo\Bar\foo的静态方法staticmethod。re
echo FOO; // resolves to constant Foo\Bar\FOO

/* 限定名称 */
subnamespace\foo(); // 解析为函数 Foo\Bar\subnamespace\foo
subnamespace\foo::staticmethod(); // 解析为类 Foo\Bar\subnamespace\f
                                  // 以及类的方法 staticmethod
echo subnamespace\FOO; // 解析为常量 Foo\Bar\subnamespace\FOO

/* 完全限定名称 */
\Foo\Bar\foo(); // 解析为函数 Foo\Bar\foo
\Foo\Bar\foo::staticmethod(); // 解析为类 Foo\Bar\foo, 以及类的方法 st
echo \Foo\Bar\FOO; // 解析为常量 Foo\Bar\FOO
?>
```

注意访问任意全局类、函数或常量，都可以使用完全限定名称，例如 \strlen() 或 \Exception 或 \INI_ALL。

在命名空间内部访问全局类、函数和常量：

```
<?php
namespace Foo;

function strlen() {}
const INI_ALL = 3;
class Exception {}

$a = \strlen('hi'); // 调用全局函数strlen
$b = \INI_ALL; // 访问全局常量 INI_ALL
$c = new \Exception('error'); // 实例化全局类 Exception
?>
```

命名空间和动态语言特征

PHP 命名空间的实现受到其语言自身的动态特征的影响。因此，如果要将下面的代码转换到命名空间中，动态访问元素。

example1.php 文件代码：

```
<?php
class classname
{
    function __construct()
    {
        echo __METHOD__,"\n";
    }
}
function funcname()
{
    echo __FUNCTION__,"\n";
}
const constname = "global";

$a = 'classname';
$obj = new $a; // prints classname::__construct
$b = 'funcname';
$b(); // prints funcname
echo constant('constname'), "\n"; // prints global
?>
```

必须使用完全限定名称（包括命名空间前缀的类名称）。注意因为在动态的类名称、函数名称或常量名称中，限定名称和完全限定名称没有区别，因此其前导的反斜杠是不必要的。

动态访问命名空间的元素

```
<?php
namespace namespacename;
class classname
{
    function __construct()
    {
        echo __METHOD__,"\n";
    }
}
function funcname()
{
    echo __FUNCTION__,"\n";
}
const constname = "namespaced";

include 'example1.php';

$a = 'classname';
$obj = new $a; // prints classname::__construct
$b = 'funcname';
$b(); // prints funcname
echo constant('constname'), "\n"; // prints global

/* note that if using double quotes, "\\namespacename\\classname" r
$a = '\namespacename\classname';
$obj = new $a; // prints namespacename\classname::__construct
$a = 'namespacename\classname';
$obj = new $a; // also prints namespacename\classname::__construct
$b = 'namespacename\funcname';
$b(); // prints namespacename\funcname
$b = '\namespacename\funcname';
$b(); // also prints namespacename\funcname
echo constant('\namespacename\constname'), "\n"; // prints namespac
echo constant('namespacename\constname'), "\n"; // also prints name
?>
```

namespace关键字和NAMESPACE常量

PHP支持两种抽象的访问当前命名空间内部元素的方法，**NAMESPACE** 魔术常量和namespace关键字。

常量**NAMESPACE**的值是包含当前命名空间名称的字符串。在全局的，不包括在任何命名空间中的代码，它包含一个空的字符串。

NAMESPACE 示例, 在命名空间中的代码

```
<?php
namespace MyProject;

echo '"', __NAMESPACE__, '"'; // 输出 "MyProject"
?>
```

NAMESPACE 示例，全局代码

```
<?php

echo '"', __NAMESPACE__, '"'; // 输出 ""
?>
```

常量 **NAMESPACE** 在动态创建名称时很有用，例如：

使用**NAMESPACE**动态创建名称

```
<?php
namespace MyProject;

function get($classname)
{
    $a = __NAMESPACE__ . '\\' . $classname;
    return new $a;
}
?>
```

关键字 namespace 可用来显式访问当前命名空间或子命名空间中的元素。它等价于类中的 self 操作符。

amespace操作符，命名空间中的代码

```
<?php
namespace MyProject;

use blah\blah as mine; // see "Using namespaces: importing/aliasing

blah\mine(); // calls function blah\blah\mine()
namespace\blah\mine(); // calls function MyProject\blah\mine()

namespace\func(); // calls function MyProject\func()
namespace\sub\func(); // calls function MyProject\sub\func()
namespace\cname::method(); // calls static method "method" of class
$a = new namespace\sub\cname(); // instantiates object of class MyP
$b = namespace\CONSTANT; // assigns value of constant MyProject\CON
?>
```

namespace操作符, 全局代码

```
<?php

namespace\func(); // calls function func()
namespace\sub\func(); // calls function sub\func()
namespace\cname::method(); // calls static method "method" of class
$a = new namespace\sub\cname(); // instantiates object of class sub
$b = namespace\CONSTANT; // assigns value of constant CONSTANT to $
?>
```

使用命名空间：别名/导入

PHP 命名空间支持 有两种使用别名或导入方式：为类名称使用别名，或为命名空间名称使用别名。注意PHP不支持导入函数或常量。

在PHP中，别名是通过操作符 use 来实现的. 下面是一个使用所有可能的三种导入方式的例子：

1、使用use操作符导入/使用别名

```
<?php
namespace foo;
use My\Full\Classname as Another;

// 下面的例子与 use My\Full\NSname as NSname 相同
use My\Full\NSname;

// 导入一个全局类
use \ArrayObject;

$obj = new namespace\Another; // 实例化 foo\Another 对象
$obj = new Another; // 实例化 My\Full\Classname  对象
NSname\subns\func(); // 调用函数 My\Full\NSname\subns\func
$a = new ArrayObject(array(1)); // 实例化 ArrayObject 对象
// 如果不使用 "use \ArrayObject" ，则实例化一个 foo\ArrayObject 对象
?>
```

2、 一行中包含多个use语句

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // 实例化 My\Full\Classname 对象
NSname\subns\func(); // 调用函数 My\Full\NSname\subns\func
?>
```

导入操作是在编译执行的，但动态的类名称、函数名称或常量名称则不是。

3、导入和动态名称

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // 实例化一个 My\Full\Classname 对象
$a = 'Another';
$obj = new $a;      // 实际化一个 Another 对象
?>
```

另外，导入操作只影响非限定名称和限定名称。完全限定名称由于是确定的，故不受导入的影响。

4、导入和完全限定名称

```
<?php
use My\Full\Classname as Another, My\Full\NSname;

$obj = new Another; // instantiates object of class My\Full\Classna
$obj = new \Another; // instantiates object of class Another
$obj = new Another\thing; // instantiates object of class My\Full\C
$obj = new \Another\thing; // instantiates object of class Another\
?>
```

使用命名空间：后备全局函数/常量

在一个命名空间中，当 PHP 遇到一个非限定的类、函数或常量名称时，它使用不同的优先策略来解析该名称。类名称总是解析到当前命名空间中的名称。因此在访问系统内部或不包含在命名空间中的类名称时，必须使用完全限定名称，例如：

1、在命名空间中访问全局类

```
<?php
namespace A\B\C;
class Exception extends \Exception {}

$a = new Exception('hi'); // $a 是类 A\B\C\Exception 的一个对象
$b = new \Exception('hi'); // $b 是类 Exception 的一个对象

$c = new ArrayObject; // 致命错误, 找不到 A\B\C\ArrayObject 类
?>
```

对于函数和常量来说，如果当前命名空间中不存在该函数或常量，PHP 会退而使用全局空间中的函数或常量。

2、 命名空间中后备的全局函数/常量

```
<?php
namespace A\B\C;

const E_ERROR = 45;
function strlen($str)
{
    return \strlen($str) - 1;
}

echo E_ERROR, "\n"; // 输出 "45"
echo INI_ALL, "\n"; // 输出 "7" - 使用全局常量 INI_ALL

echo strlen('hi'), "\n"; // 输出 "1"
if (is_array('hi')) { // 输出 "is not array"
    echo "is array\n";
} else {
    echo "is not array\n";
}
?>
```

全局空间

如果没有定义任何命名空间，所有的类与函数的定义都是在全局空间，与 PHP 引入命名空间概念前一样。在名称前加上前缀 \ 表示该名称是全局空间中的名称，即使该名称位于其它的命名空间中时也是如此。

使用全局空间说明

```
<?php
namespace A\B\C;

/* 这个函数是 A\B\C\fopen */
function fopen() {
     /* ... */
     $f = \fopen(...); // 调用全局的fopen函数
     return $f;
}
?>
```

命名空间的顺序

自从有了命名空间之后，最容易出错的该是使用类的时候，这个类的寻找路径是什么样的了。

```
<?php
namespace A;
use B\D, C\E as F;

// 函数调用

foo();      // 首先尝试调用定义在命名空间"A"中的函数foo()
            // 再尝试调用全局函数 "foo"

\foo();     // 调用全局空间函数 "foo"

my\foo();   // 调用定义在命名空间"A\my"中函数 "foo"

F();        // 首先尝试调用定义在命名空间"A"中的函数 "F"
            // 再尝试调用全局函数 "F"

// 类引用

new B();    // 创建命名空间 "A" 中定义的类 "B" 的一个对象
            // 如果未找到，则尝试自动装载类 "A\B"

new D();    // 使用导入规则，创建命名空间 "B" 中定义的类 "D" 的一个对象
            // 如果未找到，则尝试自动装载类 "B\D"

new F();    // 使用导入规则，创建命名空间 "C" 中定义的类 "E" 的一个对象
            // 如果未找到，则尝试自动装载类 "C\E"

new \B();   // 创建定义在全局空间中的类 "B" 的一个对象
            // 如果未发现，则尝试自动装载类 "B"

new \D();   // 创建定义在全局空间中的类 "D" 的一个对象
            // 如果未发现，则尝试自动装载类 "D"

new \F();   // 创建定义在全局空间中的类 "F" 的一个对象
            // 如果未发现，则尝试自动装载类 "F"

// 调用另一个命名空间中的静态方法或命名空间函数

B\foo();    // 调用命名空间 "A\B" 中函数 "foo"

B::foo();   // 调用命名空间 "A" 中定义的类 "B" 的 "foo" 方法
            // 如果未找到类 "A\B" ，则尝试自动装载类 "A\B"

D::foo();   // 使用导入规则，调用命名空间 "B" 中定义的类 "D" 的 "foo" 方法
            // 如果类 "B\D" 未找到，则尝试自动装载类 "B\D"

\B\foo();   // 调用命名空间 "B" 中的函数 "foo"

\B::foo();  // 调用全局空间中的类 "B" 的 "foo" 方法
            // 如果类 "B" 未找到，则尝试自动装载类 "B"

// 当前命名空间中的静态方法或函数
```

```
A\B::foo();    // 调用命名空间 "A\A" 中定义的类 "B" 的 "foo" 方法
               // 如果类 "A\A\B" 未找到，则尝试自动装载类 "A\A\B"

\A\B::foo();   // 调用命名空间 "A\B" 中定义的类 "B" 的 "foo" 方法
               // 如果类 "A\B" 未找到，则尝试自动装载类 "A\B"
?>
```

名称解析遵循下列规则：

1. 对完全限定名称的函数，类和常量的调用在编译时解析。例如 *new \A\B* 解析为类 *A\B*。
2. 所有的非限定名称和限定名称（非完全限定名称）根据当前的导入规则在编译时进行转换。例如，如果命名空间 *A\B\C* 被导入为 *C*，那么对 *C\D\e()* 的调用就会被转换为 *A\B\C\D\e()*。
3. 在命名空间内部，所有的没有根据导入规则转换的限定名称均会在其前面加上当前的命名空间名称。例如，在命名空间 *A\B* 内部调用 *C\D\e()*，则 *C\D\e()* 会被转换为 *A\B\C\D\e()* 。
4. 非限定类名根据当前的导入规则在编译时转换（用全名代替短的导入名称）。例如，如果命名空间 *A\B\C* 导入为C，则 *new C()* 被转换为 *new A\B\C()* 。
5. 在命名空间内部（例如A\B），对非限定名称的函数调用是在运行时解析的。例如对函数 *foo()* 的调用是这样解析的：
   i. 在当前命名空间中查找名为 *A\B\foo()* 的函数
   ii. 尝试查找并调用 全局*(global)* 空间中的函数 *foo()*。
6. 在命名空间（例如*A\B*）内部对非限定名称或限定名称类（非完全限定名称）的调用是在运行时解析的。下面是调用 *new C()* 及 *new D\E()* 的解析过程：
   *new C()*的解析:
   i. 在当前命名空间中查找*A\B\C*类。
   ii. 尝试自动装载类*A\B\C*。*new D\E()*的解析:
   iii. 在类名称前面加上当前命名空间名称变成：*A\B\D\E*，然后查找该类。
   iv. 尝试自动装载类 *A\B\D\E*。为了引用全局命名空间中的全局类，必须使用完全限定名称 *new \C()*。

PHP 表单

PHP 表单处理

PHP 超全局变量 **$_GET** 和 **$_POST** 用于收集表单数据（**form-data**）。

PHP - 一个简单的 HTML 表单

下面的例子显示了一个简单的 HTML 表单，它包含两个输入字段和一个提交按钮：

实例

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

当用户填写此表单并点击提交按钮后，表单数据会发送到名为 "welcome.php" 的 PHP 文件供处理。表单数据是通过 HTTP POST 方法发送的。

如需显示出被提交的数据，您可以简单地输出（echo）所有变量。"welcome.php" 文件是这样的：

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

输出：

```
Welcome John
Your email address is john.doe@example.com
```

使用 HTTP GET 方法也能得到相同的结果：

实例

```
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

"welcome_get.php" 是这样的：

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

上面的代码很简单。不过，最重要的内容被漏掉了。您需要对表单数据进行验证，以防止脚本出现漏洞。

注意：在处理 PHP 表单时请关注安全！

本页未包含任何表单验证程序，它只向我们展示如何发送并接收表单数据。

不过稍后的章节会为您讲解如何提高 PHP 表单的安全性！对表单适当的安全验证对于抵御黑客攻击和垃圾邮件非常重要！

GET vs. POST

GET 和 POST 都创建数组（例如，array(key => value, key2 => value2, key3 => value3, ...)）。此数组包含键/值对，其中的键是表单控件的名称，而值是来自用户的输入数据。

GET 和 POST 被视作 $_GET 和 $_POST。它们是超全局变量，这意味着对它们的访问无需考虑作用域 - 无需任何特殊代码，您能够从任何函数、类或文件访问它们。

$_GET 是通过 URL 参数传递到当前脚本的变量数组。

$_POST 是通过 HTTP POST 传递到当前脚本的变量数组。

何时使用 GET？

通过 GET 方法从表单发送的信息对任何人都是可见的（所有变量名和值都显示在 URL 中）。GET 对所发送信息的数量也有限制。限制在大于 2000 个字符。不过，由于变量显示在 URL 中，把页面添加到书签中也更为方便。

GET 可用于发送非敏感的数据。

注释：绝不能使用 GET 来发送密码或其他敏感信息！

何时使用 POST？

通过 POST 方法从表单发送的信息对其他人是不可见的（所有名称/值会被嵌入 HTTP 请求的主体中），并且对所发送信息的数量也无限制。

此外 POST 支持高阶功能，比如在向服务器上传文件时进行 multi-part 二进制输入。

不过，由于变量未显示在 URL 中，也就无法将页面添加到书签。

提示：开发者偏爱 POST 来发送表单数据。

接下来让我们看看如何安全地处理 PHP 表单！

PHP 表单验证

")

本节和下一节讲解如何使用 **PHP** 来验证表单数据。

PHP 表单验证

提示：在处理 PHP 表单时请重视安全性！

这些页面将展示如何安全地处理 PHP 表单。对 HTML 表单数据进行适当的验证对于防范黑客和垃圾邮件很重要！

我们稍后使用的 HTML 表单包含多种输入字段：必需和可选的文本字段、单选按钮以及提交按钮：

```
<h2>PHP 表单验证实例</h2>
<p><span class="error">* 必填字段</span></p>

<form method="post" action="/demo/demo_form_validation_complete.php

姓名：
<input type="text" name="name" value="">
<span class="error">* </span>
<br><br>
电邮：
<input type="text" name="email" value="">
<span class="error">* </span>
<br><br>
网址：
<input type="text" name="website" value="">
<span class="error"></span>
<br><br>
<label>
评论：
<textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
性别：
<input type="radio" name="gender"  value="female">女性
<input type="radio" name="gender"  value="male">男性
<span class="error">* </span>
<br><br>
<input type="submit" name="submit" value="提交">

</form>

<h2>您的输入：</h2>
```

上面的表单使用如下验证规则：

字段	验证规则
Name	必需。必须包含字母和空格。
E-mail	必需。必须包含有效的电子邮件地址（包含 @ 和 .）。
Website	可选。如果选填，则必须包含有效的 URL。
Comment	可选。多行输入字段（文本框）。
Gender	必需。必须选择一项。

首先我们看一下这个表单的纯 HTML 代码：

文本字段

name、email 和 website 属于文本输入元素，comment 字段是文本框。HTML 代码是这样的：

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

单选按钮

gender 字段是单选按钮，HTML 代码是这样的：

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
```

表单元素

表单的 HTML 代码是这样的：

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["
```

当提交此表单时，通过 method="post" 发送表单数据。

什么是 **$_SERVER["PHP_SELF"]** 变量？

$_SERVER["PHP_SELF"] 是一种超全局变量，它返回当前执行脚本的文件名。

因此，$_SERVER["PHP_SELF"] 将表单数据发送到页面本身，而不是跳转到另一张页面。这样，用户就能够在表单页面获得错误提示信息。

什么是 **htmlspecialchars()** 函数？

htmlspecialchars() 函数把特殊字符转换为 HTML 实体。这意味着 < 和 > 之类的 HTML 字符会被替换为 < 和 > 。这样可防止攻击者通过在表单中注入 HTML 或 JavaScript 代码（跨站点脚本攻击）对代码进行利用。

关于 **PHP** 表单安全性的重要提示

$_SERVER["PHP_SELF"] 变量能够被黑客利用！

如果您的页面使用了 PHP_SELF，用户能够输入下划线然后执行跨站点脚本（XSS）。

提示：跨站点脚本（Cross-site scripting，XSS）是一种计算机安全漏洞类型，常见于 Web 应用程序。XSS 能够使攻击者向其他用户浏览的网页中输入客户端脚本。

假设我们的一张名为 "test_form.php" 的页面中有如下表单：

```
<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">
```

现在，如果用户进入的是地址栏中正常的 URL："http://www.example.com/test_form.php"，上面的代码会转换为：

```
<form method="post" action="test_form.php">
```

到目前，一切正常。

不过，如果用户在地址栏中键入了如下 URL：

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacke
```

在这种情况下，上面的代码会转换为：

```
<form method="post" action="test_form.php"/><script>alert('hacked')
```

这段代码加入了一段脚本和一个提示命令。并且当此页面加载后，就会执行 JavaScript 代码（用户会看到一个提示框）。这仅仅是一个关于 PHP_SELF 变量如何被利用的简单无害案例。

您应该意识到 *<script>* 标签内能够添加任何 *JavaScript* 代码！黑客能够把用户重定向到另一台服务器上的某个文件，该文件中的恶意代码能够更改全局变量或将表单提交到其他地址以保存用户数据，等等。

如果避免 $_SERVER["PHP_SELF"] 被利用？

通过使用 htmlspecialchars() 函数能够避免 $_SERVER["PHP_SELF"] 被利用。

表单代码是这样的：

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["I
```

htmlspecialchars() 函数把特殊字符转换为 HTML 实体。现在，如果用户试图利用 PHP_SELF 变量，会导致如下输出：

```
<form method="post" action="test_form.php/"><script>alert('hacked')
```

无法利用，没有危害！

通过 PHP 验证表单数据

我们要做的第一件事是通过 PHP 的 htmlspecialchars() 函数传递所有变量。

在我们使用 htmlspecialchars() 函数后，如果用户试图在文本字段中提交以下内容：

```
<script>location.href('http://www.hacked.com')</script>
```

- 代码不会执行，因为会被保存为转义代码，就像这样：

```
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
```

现在这条代码显示在页面上或 e-mail 中是安全的。

在用户提交该表单时，我们还要做两件事：

1. （通过 PHP trim() 函数）去除用户输入数据中不必要的字符（多余的空格、制表符、换行）
2. （通过 PHP stripslashes() 函数）删除用户输入数据中的反斜杠（\）

接下来我们创建一个检查函数（相比一遍遍地写代码，这样效率更好）。

我们把函数命名为 test_input()。

现在，我们能够通过 test_input() 函数检查每个 $_POST 变量，脚本是这样的：

实例

```
<?php
// 定义变量并设置为空值
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

请注意在脚本开头，我们检查表单是否使用 $_SERVER["REQUEST_METHOD"] 进行提交。如果 REQUEST_METHOD 是 POST，那么表单已被提交 - 并且应该对其进行验证。如果未提交，则跳过验证并显示一个空白表单。

不过，在上面的例子中，所有输入字段都是可选的。即使用户未输入任何数据，脚本也能正常工作。

下一步是制作必填输入字段，并创建需要时使用的错误消息。

")

PHP 表单验证 - 必填字段

本节展示如何制作必填输入字段，并创建需要时所用的错误消息。

PHP - 输入字段

从上一节中的验证规则中，我们看到 "Name", "E-mail" 以及 "Gender" 字段是必需的。这些字段不能为空且必须在 HTML 表单中填写。

字段	验证规则
Name	必需。必须包含字母和空格。
E-mail	必需。必须包含有效的电子邮件地址（包含 @ 和 .）。
Website	可选。如果选填，则必须包含有效的 URL。
Comment	可选。多行输入字段（文本框）。
Gender	必需。必须选择一项。

在上一节中，所有输入字段都是可选的。

在下面的代码中我们增加了一些新变量：$nameErr、$emailErr、$genderErr 以及 $websiteErr。这些错误变量会保存被请求字段的错误消息。我们还为每个 $_POST 变量添加了一个 if else 语句。这条语句检查 $_POST 变量是否为空（通过 PHP empty() 函数）。如果为空，则错误消息会存储于不同的错误变量中。如果不为空，则通过 test_input() 函数发送用户输入数据：

```
<?php
// 定义变量并设置为空值
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP - 显示错误消息

在 HTML 表单中，我们在每个被请求字段后面增加了一点脚本。如果需要，会生成恰当的错误消息（如果用户未填写必填字段就试图提交表单）：

实例

```
<form method="post" action="<?php echo htmlspecialchars($_SERVER["F

Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
<label>Comment: <textarea name="comment" rows="5" cols="40"></texta
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">

</form>
```

接下来是验证输入数据，即“Name 字段是否只包含字母和空格？”，以及“E-mail 字段是否包含有效的电子邮件地址语法？”，并且如果填写了 Website 字段，“这个字段是否包含了有效的 URL？”。

PHP 表单验证 - 验证 E-mail 和 URL

本节展示如何验证名字、电邮和 **URL**。

PHP - 验证名字

以下代码展示的简单方法检查 name 字段是否包含字母和空格。如果 name 字段无效，则存储一条错误消息：

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "只允许字母和空格！";
}
```

注释：preg_match() 函数检索字符串的模式，如果模式存在则返回 true，否则返回 false。

PHP - 验证 E-mail

以下代码展示的简单方法检查 e-mail 地址语法是否有效。如果无效则存储一条错误消息：

```
$email = test_input($_POST["email"]);
if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email)) {
  $emailErr = "无效的 email 格式！";
}
```

PHP - 验证 URL

以下代码展示的方法检查 URL 地址语法是否有效（这条正则表达式同时允许 URL 中的斜杠）。如果 URL 地址语法无效，则存储一条错误消息：

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=-
=~_|]/i",$website)) {
  $websiteErr = "无效的 URL";
}
```

PHP - 验证 Name、E-mail、以及 URL

现在，脚本是这样的：

实例

```
<?php
// 定义变量并设置为空值
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // 检查名字是否包含字母和空格
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // 检查电邮地址语法是否有效
    if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // 检查 URL 地址语言是否有效（此正则表达式同样允许 URL 中的下划线）
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/
    =~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

接下来向您讲解如何防止表单在用户提交表单后清空所有输入字段。

PHP 表单验证 - 完成表单实例

本节展示如何在用户提交表单后保留输入字段中的值。

PHP - 保留表单中的值

如需在用户点击提交按钮后在输入字段中显示值，我们在以下输入字段的 value 属性中增加了一小段 PHP 脚本：name、email 以及 website。在 comment 文本框字段中，我们把脚本放到了 <textarea> 与 </textarea> 之间。这些脚本输出 $name、$email、$website 和 $comment 变量的值。

然后，我们还需要显示选中了哪个单选按钮。对此，我们必须操作 checked 属性（而非单选按钮的 value 属性）：

```
Name: <input type="text" name="name" value="<?php echo $name;?>">

E-mail: <input type="text" name="email" value="<?php echo $email;?>

Website: <input type="text" name="website" value="<?php echo $websi

Comment: <textarea name="comment" rows="5" cols="40"><?php echo $co

Gender:

<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
```

PHP - 完整的表单实例

下面是 PHP 表单验证实例的完整代码：

实例

```
<h2>PHP 表单验证实例</h2>
<p><span class="error">* 必填字段</span></p>

<form method="post" action="/demo/demo_form_validation_complete.php

姓名：
<input type="text" name="name" value="">
<span class="error">* </span>
<br><br>
电邮：
<input type="text" name="email" value="">
<span class="error">* </span>
<br><br>
网址：
<input type="text" name="website" value="">
<span class="error"></span>
<br><br>
<label>
评论：
<textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
性别：
<input type="radio" name="gender"  value="female">女性
<input type="radio" name="gender"  value="male">男性
<span class="error">* </span>
<br><br>
<input type="submit" name="submit" value="提交">

</form>

<h2>您的输入：</h2>
```

PHP $_GET 变量

在 PHP 中，预定义的 $_GET 变量用于收集来自 method="get" 的表单中的值。

$_GET 变量

预定义的 $_GET 变量用于收集来自 method="get" 的表单中的值。

从带有 GET 方法的表单发送的信息，对任何人都是可见的（会显示在浏览器的地址栏），并且对发送信息的量也有限制。

实例

```
<form action="welcome.php" method="get">
Name: <input type="text" name="fname">
Age: <input type="text" name="age">
<input type="submit">
</form>
```

当用户点击 "Submit" 按钮时，发送到服务器的 URL 如下所示：

http://www.w3cschool.cc/welcome.php?fname=Peter&age=37

"welcome.php" 文件现在可以通过 $_GET 变量来收集表单数据了（请注意，表单域的名称会自动成为 $_GET 数组中的键）：

```
Welcome <?php echo $_GET["fname"]; ?>.<br>
You are <?php echo $_GET["age"]; ?> years old!
```

何时使用 method="get"？

在 HTML 表单中使用 method="get" 时，所有的变量名和值都会显示在 URL 中。

注释：所以在发送密码或其他敏感信息时，不应该使用这个方法！

然而，正因为变量显示在 URL 中，因此可以在收藏夹中收藏该页面。在某些情况下，这是很有用的。

注释：HTTP GET 方法不适合大型的变量值。它的值是不能超过 2000 个字符的。

PHP $_POST 变量

在 PHP 中，预定义的 $_POST 变量用于收集来自 method="post" 的表单中的值。

$_POST 变量

预定义的 $_POST 变量用于收集来自 method="post" 的表单中的值。

从带有 POST 方法的表单发送的信息，对任何人都是不可见的（不会显示在浏览器的地址栏），并且对发送信息的量也没有限制。

注释：然而，默认情况下，POST 方法的发送信息的量最大值为 8 MB（可通过设置 php.ini 文件中的 post_max_size 进行更改）。

实例

```
<form action="welcome.php" method="post">
Name: <input type="text" name="fname">
Age: <input type="text" name="age">
<input type="submit">
</form>
```

当用户点击 "Submit" 按钮时，URL 如下所示：

http://www.w3cschool.cc/welcome.php

"welcome.php" 文件现在可以通过 $_POST 变量来收集表单数据了（请注意，表单域的名称会自动成为 $_POST 数组中的键）：

```
Welcome <?php echo $_POST["fname"]; ?>!<br>
You are <?php echo $_POST["age"]; ?> years old.
```

何时使用 method="post"？

从带有 POST 方法的表单发送的信息，对任何人都是不可见的，并且对发送信息的量也没有限制。

然而，由于变量不显示在 URL 中，所以无法把页面加入书签。

PHP $_REQUEST 变量

预定义的 $_REQUEST 变量包含了 $_GET、$_POST 和 $_COOKIE 的内容。

$_REQUEST 变量可用来收集通过 GET 和 POST 方法发送的表单数据。

实例

```
Welcome <?php echo $_REQUEST["fname"]; ?>!<br>
You are <?php echo $_REQUEST["age"]; ?> years old.
```

PHP 高级

PHP 多维数组

")

在本教程之前的章节中，我们已经知道数组是一种数/值对的简单列表。

不过，有时您希望用一个以上的键存储值。

可以用多维数组进行存储。

PHP - 多维数组

多维数组指的是包含一个或多个数组的数组。

PHP 能理解两、三、四或五级甚至更多级的多维数组。不过，超过三级深的数组对于大多数人难于管理。

注释：数组的维度指示您需要选择元素的索引数。

- 对于二维数组，您需要两个索引来选取元素
- 对于三维数组，您需要三个索引来选取元素

PHP - 两维数组

两维数组是数组的数组（三维数组是数组的数组的数组）。

首先，让我们看看下面的表格：

品牌	库存	销量
Volvo	33	20
BMW	17	15
Saab	5	2
Land Rover	15	11

我们能够在两维数组中存储上表中的数据，就像这样：

```
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );
```

现在这个两维数组包含了四个数组，并且它有两个索引（下标）：行和列。

如需访问 $cars 数组中的元素，我们必须使用两个索引（行和列）：

实例

```
<?php
echo $cars[0][0].": 库存：".$cars[0][1].", 销量：".$cars[0][2].".<br>
echo $cars[1][0].": 库存：".$cars[1][1].", 销量：".$cars[1][2].".<br>
echo $cars[2][0].": 库存：".$cars[2][1].", 销量：".$cars[2][2].".<br>
echo $cars[3][0].": 库存：".$cars[3][1].", 销量：".$cars[3][2].".<br>
?>
```

我们也可以在 For 循环中使用另一个 For 循环，来获得 $cars 数组中的元素（我们仍需使用两个索引）：

实例

```
<?php
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

")

PHP 日期和时间

PHP date() 函数用于对日期或时间进行格式化。

PHP Date() 函数

PHP Date() 函数把时间戳格式化为更易读的日期和时间。

语法

```
date(format,timestamp)
```

参数	描述
format	必需。规定时间戳的格式。
timestamp	可选。规定时间戳。默认是当前时间和日期。

注释：时间戳是一种字符序列，它表示具体事件发生的日期和事件。

获得简单的日期

date() 函数的格式参数是必需的，它们规定如何格式化日期或时间。

下面列出了一些常用于日期的字符：

- d - 表示月里的某天（01-31）
- m - 表示月（01-12）
- Y - 表示年（四位数）
- 1 - 表示周里的某天

其他字符，比如 "/", "." 或 "-" 也可被插入字符中，以增加其他格式。

下面的例子用三种不同方法格式今天的日期：

实例

```
<?php
echo "今天是 " . date("Y/m/d") . "<br>";
echo "今天是 " . date("Y.m.d") . "<br>";
echo "今天是 " . date("Y-m-d") . "<br>";
echo "今天是 " . date("l");
?>
```

PHP 提示 - 自动版权年份

使用 date() 函数在您的网站上自动更新版本年份：

实例

```
? 2010-<?php echo date("Y")?>
```

获得简单的时间

下面是常用于时间的字符：

- h - 带有首位零的 12 小时小时格式
- i - 带有首位零的分钟
- s - 带有首位零的秒（00 -59）
- a - 小写的午前和午后（am 或 pm）

下面的例子以指定的格式输出当前时间：

实例

```
<?php
echo "现在时间是 " . date("h:i:sa");
?>
```

注释：请注意 PHP date() 函数会返回服务器的当前日期/时间！

获得时区

如果从代码返回的不是正确的时间，有可能是因为您的服务器位于其他国家或者被设置为不同时区。

因此，如果您需要基于具体位置的准确时间，您可以设置要用的时区。

下面的例子把时区设置为 "Asia/Shanghai"，然后以指定格式输出当前时间：

实例

```
<?php
date_default_timezone_set("Asia/Shanghai");
echo "当前时间是 " . date("h:i:sa");
?>
```

通过 PHP mktime() 创建日期

date() 函数中可选的时间戳参数规定时间戳。如果您未规定时间戳，将使用当前日期和时间（正如上例中那样）。

mktime() 函数返回日期的 Unix 时间戳。Unix 时间戳包含 Unix 纪元（1970 年 1 月 1 日 00:00:00 GMT）与指定时间之间的秒数。

语法

```
mktime(hour,minute,second,month,day,year)
```

下面的例子使用 mktime() 函数中的一系列参数来创建日期和时间：

实例

```
<?php
$d=mktime(9, 12, 31, 6, 10, 2015);
echo "创建日期是 " . date("Y-m-d h:i:sa", $d);
?>
```

通过 PHP strtotime() 用字符串来创建日期

PHP strtotime() 函数用于把人类可读的字符串转换为 Unix 时间。

语法

```
strtotime(time,now)
```

下面的例子通过 strtotime() 函数创建日期和时间：

实例

```
<?php
$d=strtotime("10:38pm April 15 2015");
echo "创建日期是 " . date("Y-m-d h:i:sa", $d);
?>
```

PHP 在将字符串转换为日期这方面非常聪明，所以您能够使用各种值：

实例

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

不过，strtotime() 并不完美，所以请记得检查放入其中的字符串。

更多日期实例

下例输出下周六的日期：

实例

```
<?php
$startdate = strtotime("Saturday");
$enddate = strtotime("+6 weeks",$startdate);

while ($startdate < $enddate) {
  echo date("M d", $startdate),"<br>";
  $startdate = strtotime("+1 week", $startdate);
}
?>
```

下例输出七月四日之前的天数：

实例

```
<?php
$d1=strtotime("December 31");
$d2=ceil(($d1-time())/60/60/24);
echo "距离十二月三十一日还有：" . $d2 ." 天。";
?>
```

完整的 PHP 日期参考手册

如需所有日期函数的完整手册，请访问我们的 PHP 日期参考手册。

该手册包含每个函数的简要描述以及使用示例。

PHP Include 文件

")

服务器端包含 **(SSI)** 用于创建可在多个页面重复使用的函数、页眉、页脚或元素。

include （或 require）语句会获取指定文件中存在的所有文本/代码/标记，并复制到使用 include 语句的文件中。

包含文件很有用，如果您需要在网站的多张页面上引用相同的 PHP、HTML 或文本的话。

PHP include 和 require 语句

通过 include 或 require 语句，可以将 PHP 文件的内容插入另一个 PHP 文件（在服务器执行它之前）。

include 和 require 语句是相同的，除了错误处理方面：

- require 会生成致命错误（E_COMPILE_ERROR）并停止脚本
- include 只生成警告（E_WARNING），并且脚本会继续

因此，如果您希望继续执行，并向用户输出结果，即使包含文件已丢失，那么请使用 include。否则，在框架、CMS 或者复杂的 PHP 应用程序编程中，请始终使用 require 向执行流引用关键文件。这有助于提高应用程序的安全性和完整性，在某个关键文件意外丢失的情况下。

包含文件省去了大量的工作。这意味着您可以为所有页面创建标准页头、页脚或者菜单文件。然后，在页头需要更新时，您只需更新这个页头包含文件即可。

语法

```
include 'filename';
```

或

```
require 'filename';
```

PHP include 实例

例子 1

假设我们有一个名为 "footer.php" 的标准的页脚文件，就像这样：

```
<?php
echo "<p>Copyright ? 2006-" . date("Y") . " W3School.com.cn</p>";
?>
```

如需在一张页面中引用这个页脚文件，请使用 include 语句：

```
<html>
<body>

<h1>欢迎访问我们的首页！</h1>
<p>一段文本。</p>
<p>一段文本。</p>
<?php include 'footer.php';?>

</body>
</html>
```

例子 2

假设我们有一个名为 "menu.php" 的标准菜单文件：

```
<?php
echo '<a href="/index.asp">首页</a> -
<a href="/html/index.asp">HTML 教程</a> -
<a href="/css/index.asp">CSS 教程</a> -
<a href="/js/index.asp">JavaScript 教程</a> -
<a href="/php/index.asp">PHP 教程</a>';
?>
```

网站中的所有页面均使用此菜单文件。具体的做法是（我们使用了一个 <div> 元素，这样今后就可以轻松地通过 CSS 设置样式）：

```
<html>
<body>

<div class="menu">
<?php include 'menu.php';?>
</div>

<h1>欢迎访问我的首页！</h1>
<p>Some text.</p>
<p>Some more text.</p>

</body>
</html>
```

例子 3

假设我们有一个名为 "vars.php" 的文件，其中定义了一些变量：

```
<?php
$color='银色的';
$car='奔驰轿车';
?>
```

然后，如果我们引用这个 "vars.php" 文件，就可以在调用文件中使用这些变量：

```
<html>
<body>

<h1>欢迎访问我的首页！</h1>
<?php
include 'vars.php';
echo "我有一辆" . $color . $car "。";
?>

</body>
</html>
```

PHP include vs. require

require 语句同样用于向 PHP 代码中引用文件。

不过，include 与 require 有一个巨大的差异：如果用 include 语句引用某个文件并且 PHP 无法找到它，脚本会继续执行：

实例

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php
include 'noFileExists.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

如果我们使用 require 语句完成相同的案例，echo 语句不会继续执行，因为在 require 语句返回严重错误之后脚本就会终止执行：

实例

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php
require 'noFileExists.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

注释：

请在此时使用 require：当文件被应用程序请求时。

请在此时使用 include：当文件不是必需的，且应用程序在文件未找到时应该继续运行时。

")

PHP 文件处理

PHP 操作文件

PHP 拥有的多种函数可供创建、读取、上传以及编辑文件。

注意：请谨慎操作文件！

当您操作文件时必须非常小心。如果您操作失误，可能会造成非常严重的破坏。常见的错误是：

- 编辑错误的文件
- 被垃圾数据填满硬盘
- 意外删除文件内容

PHP readfile() 函数

readfile() 函数读取文件，并把它写入输出缓冲。

假设我们有一个名为 "webdictionary.txt" 的文本文件，存放在服务器上，就像这样：

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

读取此文件并写到输出流的 PHP 代码如下（如读取成功则 readfile() 函数返回字节数）：

实例

```
<?php
echo readfile("webdictionary.txt");
?>
```

如果您想做的所有事情就是打开一个文件并读取器内容，那么 readfile() 函数很有用。

下一节会讲解更多有关文件处理的内容。

PHP Filesystem 参考手册

如需完整的 PHP 文件系统参考手册，请访问 W3School 提供的 PHP Filesystem 参考手册。

PHP 文件打开/读取/读取

在本节中，我们向您讲解如何在服务器上打开、读取以及关闭文件。

PHP Open File - fopen()

打开文件的更好的方法是通过 fopen() 函数。此函数为您提供比 readfile() 函数更多的选项。

在课程中，我们将使用文本文件 "webdictionary.txt"：

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

fopen() 的第一个参数包含被打开的文件名，第二个参数规定打开文件的模式。如果 fopen() 函数未能打开指定的文件，下面的例子会生成一段消息：

实例

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open fi
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

提示：我们接下来将学习 fread() 以及 fclose() 函数。

文件会以如下模式之一打开：

模式	描述
r	打开文件为只读。文件指针在文件的开头开始。
w	打开文件为只写。删除文件的内容或创建一个新的文件，如果它不存在。文件指针在文件的开头开始。
a	打开文件为只写。文件中的现有数据会被保留。文件指针在文件结尾开始。创建新的文件，如果文件不存在。
x	创建新文件为只写。返回 FALSE 和错误，如果文件已存在。
r+	打开文件为读/写、文件指针在文件开头开始。
w+	打开文件为读/写。删除文件内容或创建新文件，如果它不存在。文件指针在文件开头开始。
a+	打开文件为读/写。文件中已有的数据会被保留。文件指针在文件结尾开始。创建新文件，如果它不存在。
x+	创建新文件为读/写。返回 FALSE 和错误，如果文件已存在。

PHP 读取文件 - fread()

fread() 函数读取打开的文件。

fread() 的第一个参数包含待读取文件的文件名，第二个参数规定待读取的最大字节数。

如下 PHP 代码把 "webdictionary.txt" 文件读至结尾：

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP 关闭文件 - fclose()

fclose() 函数用于关闭打开的文件。

注释：用完文件后把它们全部关闭是一个良好的编程习惯。您并不想打开的文件占用您的服务器资源。

fclose() 需要待关闭文件的名称（或者存有文件名的变量）：

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP 读取单行文件 - fgets()

fgets() 函数用于从文件读取单行。

下例输出 "webdictionary.txt" 文件的首行：

实例

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open fi
echo fgets($myfile);
fclose($myfile);
?>
```

注释：调用 fgets() 函数之后，文件指针会移动到下一行。

PHP 检查 End-Of-File - feof()

feof() 函数检查是否已到达 "end-of-file" (EOF)。

feof() 对于遍历未知长度的数据很有用。

下例逐行读取 "webdictionary.txt" 文件，直到 end-of-file：

实例

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open fi
// 输出单行直到 end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

PHP 读取单字符 - fgetc()

fgetc() 函数用于从文件中读取单个字符。

下例逐字符读取 "webdictionary.txt" 文件，直到 end-of-file：

实例

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open fi
// 输出单字符直到 end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

注释：在调用 fgetc() 函数之后，文件指针会移动到下一个字符。

PHP Filesystem 参考手册

如需完整的 PHP 文件系统参考手册，请访问 W3School 提供的 PHP Filesystem 参考手册。

PHP 文件创建/写入

在本节中，我们将为您讲解如何在服务器上创建并写入文件。

PHP 创建文件 - fopen()

fopen() 函数也用于创建文件。也许有点混乱，但是在 PHP 中，创建文件所用的函数与打开文件的相同。

如果您用 fopen() 打开并不存在的文件，此函数会创建文件，假定文件被打开为写入（w）或增加（a）。

下面的例子创建名为 "testfile.txt" 的新文件。此文件将被创建于 PHP 代码所在的相同目录中：

实例

```
$myfile = fopen("testfile.txt", "w")
```

PHP 文件权限

如果您试图运行这段代码时发生错误，请检查您是否有向硬盘写入信息的 PHP 文件访问权限。

PHP 写入文件 - fwrite()

fwrite() 函数用于写入文件。

fwrite() 的第一个参数包含要写入的文件的文件名，第二个参数是被写的字符串。

下面的例子把姓名写入名为 "newfile.txt" 的新文件中：

实例

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Bill Gates\n";
fwrite($myfile, $txt);
$txt = "Steve Jobs\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

请注意，我们向文件 "newfile.txt" 写了两次。在每次我们向文件写入时，在我们发送的字符串 $txt 中，第一次包含 "Bill Gates"，第二次包含 "Steve Jobs"。在写入完成后，我们使用 fclose() 函数来关闭文件。

如果我们打开 "newfile.txt" 文件，它应该是这样的：

```
Bill Gates
Steve Jobs
```

PHP 覆盖（Overwriting）

如果现在 "newfile.txt" 包含了一些数据，我们可以展示在写入已有文件时发生的的事情。所有已存在的数据会被擦除并以一个新文件开始。

在下面的例子中，我们打开一个已存在的文件 "newfile.txt"，并向其中写入了一些新数据：

实例

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

如果现在我们打开这个 "newfile.txt" 文件，Bill 和 Steve 都已消失，只剩下我们刚写入的数据：

```
Mickey Mouse
Minnie Mouse
```

PHP Filesystem 参考手册

如需完整的 PHP 文件系统参考手册，请访问 W3School 提供的 PHP Filesystem 参考手册。

PHP 文件上传

通过 **PHP**，可以把文件上传到服务器。

创建一个文件上传表单

允许用户从表单上传文件是非常有用的。

请看下面这个供上传文件的 HTML 表单：

```
<html>
<body>

<form action="upload_file.php" method="post"
enctype="multipart/form-data">
<label for="file">Filename:</label>
<input type="file" name="file" id="file" />
<br />
<input type="submit" name="submit" value="Submit" />
</form>

</body>
</html>
```

请留意如下有关此表单的信息：

<form> 标签的 enctype 属性规定了在提交表单时要使用哪种内容类型。在表单需要二进制数据时，比如文件内容，请使用 "multipart/form-data"。

<input> 标签的 type="file" 属性规定了应该把输入作为文件来处理。举例来说，当在浏览器中预览时，会看到输入框旁边有一个浏览按钮。

注释：允许用户上传文件是一个巨大的安全风险。请仅仅允许可信的用户执行文件上传操作。

创建上传脚本

"upload_file.php" 文件含有供上传文件的代码：

```
<?php
if ($_FILES["file"]["error"] > 0)
  {
  echo "Error: " . $_FILES["file"]["error"] . "<br />";
  }
else
  {
  echo "Upload: " . $_FILES["file"]["name"] . "<br />";
  echo "Type: " . $_FILES["file"]["type"] . "<br />";
  echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />";
  echo "Stored in: " . $_FILES["file"]["tmp_name"];
  }
?>
```

通过使用 PHP 的全局数组 $_FILES，你可以从客户计算机向远程服务器上传文件。

第一个参数是表单的 input name，第二个下标可以是 "name", "type", "size", "tmp_name" 或 "error"。就像这样：

- $_FILES["file"]["name"] - 被上传文件的名称
- $_FILES["file"]["type"] - 被上传文件的类型
- $_FILES["file"]["size"] - 被上传文件的大小，以字节计
- $_FILES["file"]["tmp_name"] - 存储在服务器的文件的临时副本的名称
- $_FILES["file"]["error"] - 由文件上传导致的错误代码

这是一种非常简单文件上传方式。基于安全方面的考虑，您应当增加有关什么用户有权上传文件的限制。

上传限制

在这个脚本中，我们增加了对文件上传的限制。用户只能上传 .gif 或 .jpeg 文件，文件大小必须小于 20 kb：

```
<?php

if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/pjpeg"))
&& ($_FILES["file"]["size"] < 20000))
  {
  if ($_FILES["file"]["error"] > 0)
    {
    echo "Error: " . $_FILES["file"]["error"] . "<br />";
    }
  else
    {
    echo "Upload: " . $_FILES["file"]["name"] . "<br />";
    echo "Type: " . $_FILES["file"]["type"] . "<br />";
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />"
    echo "Stored in: " . $_FILES["file"]["tmp_name"];
    }
  }
else
  {
  echo "Invalid file";
  }

?>
```

注释：对于 IE，识别 jpg 文件的类型必须是 pjpeg，对于 FireFox，必须是 jpeg。

保存被上传的文件

上面的例子在服务器的 PHP 临时文件夹创建了一个被上传文件的临时副本。

这个临时的复制文件会在脚本结束时消失。要保存被上传的文件，我们需要把它拷贝到另外的位置：

```
<?php
if ((($_FILES["file"]["type"] == "image/gif")
|| ($_FILES["file"]["type"] == "image/jpeg")
|| ($_FILES["file"]["type"] == "image/pjpeg"))
&& ($_FILES["file"]["size"] < 20000))
  {
  if ($_FILES["file"]["error"] > 0)
    {
    echo "Return Code: " . $_FILES["file"]["error"] . "<br />";
    }
  else
    {
    echo "Upload: " . $_FILES["file"]["name"] . "<br />";
    echo "Type: " . $_FILES["file"]["type"] . "<br />";
    echo "Size: " . ($_FILES["file"]["size"] / 1024) . " Kb<br />",
    echo "Temp file: " . $_FILES["file"]["tmp_name"] . "<br />";

    if (file_exists("upload/" . $_FILES["file"]["name"]))
      {
      echo $_FILES["file"]["name"] . " already exists. ";
      }
    else
      {
      move_uploaded_file($_FILES["file"]["tmp_name"],
      "upload/" . $_FILES["file"]["name"]);
      echo "Stored in: " . "upload/" . $_FILES["file"]["name"];
      }
    }
  }
else
  {
  echo "Invalid file";
  }
?>
```

上面的脚本检测了是否已存在此文件，如果不存在，则把文件拷贝到指定的文件夹。

注释：这个例子把文件保存到了名为 "upload" 的新文件夹。

PHP Cookies

cookie 常用于识别用户。

什么是 **Cookie**？

cookie 常用于识别用户。cookie 是服务器留在用户计算机中的小文件。每当相同的计算机通过浏览器请求页面时，它同时会发送 cookie。通过 PHP，您能够创建并取回 cookie 的值。

如何创建 **cookie**？

setcookie() 函数用于设置 cookie。

注释：setcookie() 函数必须位于 <html> 标签之前。

语法

```
setcookie(name, value, expire, path, domain);
```

例子

在下面的例子中，我们将创建名为 "user" 的 cookie，把为它赋值 "Alex Porter"。我们也规定了此 cookie 在一小时后过期：

```
<?php
setcookie("user", "Alex Porter", time()+3600);
?>

<html>
<body>

</body>
</html>
```

注释：在发送 cookie 时，cookie 的值会自动进行 URL 编码，在取回时进行自动解码（为防止 URL 编码，请使用 setrawcookie() 取而代之）。

如何取回 **Cookie** 的值？

PHP 的 $_COOKIE 变量用于取回 cookie 的值。

在下面的例子中，我们取回了名为 "user" 的 cookie 的值，并把它显示在了页面上：

```
<?php
// Print a cookie
echo $_COOKIE["user"];

// A way to view all cookies
print_r($_COOKIE);
?>
```

在下面的例子中，我们使用 isset() 函数来确认是否已设置了 cookie：

```
<html>
<body>

<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br />";
else
  echo "Welcome guest!<br />";
?>

</body>
</html>
```

如何删除 cookie？

当删除 cookie 时，您应当使过期日期变更为过去的时间点。

删除的例子：

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time()-3600);
?>
```

如果浏览器不支持 cookie 该怎么办？

如果您的应用程序涉及不支持 cookie 的浏览器，您就不得不采取其他方法在应用程序中从一张页面向另一张页面传递信息。一种方式是从表单传递数据（有关表单和用户输入的内容，稍早前我们已经在本教程中介绍过了）。

下面的表单在用户单击提交按钮时向 "welcome.php" 提交了用户输入：

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>

</body>
</html>
```

取回 "welcome.php" 中的值，就像这样：

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old.

</body>
</html>
```

PHP Sessions

PHP session 变量用于存储有关用户会话的信息，或更改用户会话的设置。**Session** 变量保存的信息是单一用户的，并且可供应用程序中的所有页面使用。

PHP Session 变量

当您运行一个应用程序时，您会打开它，做些更改，然后关闭它。这很像一次会话。计算机清楚你是谁。它知道你何时启动应用程序，并在何时终止。但是在因特网上，存在一个问题：服务器不知道你是谁以及你做什么，这是由于 HTTP 地址不能维持状态。

通过在服务器上存储用户信息以便随后使用，PHP session 解决了这个问题（比如用户名称、购买商品等）。不过，会话信息是临时的，在用户离开网站后将被删除。如果您需要永久储存信息，可以把数据存储在数据库中。

Session 的工作机制是：为每个访问者创建一个唯一的 id (UID)，并基于这个 UID 来存储变量。UID 存储在 cookie 中，亦或通过 URL 进行传导。

开始 PHP Session

在您把用户信息存储到 PHP session 中之前，首先必须启动会话。

注释：session_start() 函数必须位于 <html> 标签之前：

```
<?php session_start(); ?>

<html>
<body>

</body>
</html>
```

上面的代码会向服务器注册用户的会话，以便您可以开始保存用户信息，同时会为用户会话分配一个 UID。

存储 Session 变量

存储和取回 session 变量的正确方法是使用 PHP $_SESSION 变量：

```
<?php
session_start();
// store session data
$_SESSION['views']=1;
?>

<html>
<body>

<?php
//retrieve session data
echo "Pageviews=". $_SESSION['views'];
?>

</body>
</html>
```

输出：

```
Pageviews=1
```

在下面的例子中，我们创建了一个简单的 page-view 计数器。isset() 函数检测是否已设置 "views" 变量。如果已设置 "views" 变量，我们累加计数器。如果 "views" 不存在，则我们创建 "views" 变量，并把它设置为 1：

```
<?php
session_start();

if(isset($_SESSION['views']))
  $_SESSION['views']=$_SESSION['views']+1;

else
  $_SESSION['views']=1;
echo "Views=". $_SESSION['views'];
?>
```

终结 Session

如果您希望删除某些 session 数据，可以使用 unset() 或 session_destroy() 函数。

unset() 函数用于释放指定的 session 变量：

```
<?php
unset($_SESSION['views']);
?>
```

您也可以通过 session_destroy() 函数彻底终结 session：

```
<?php
session_destroy();
?>
```

注释：session_destroy() 将重置 session，您将失去所有已存储的 session 数据。

PHP 发送电子邮件

PHP 允许您从脚本直接发送电子邮件。

PHP mail() 函数

PHP mail() 函数用于从脚本中发送电子邮件。

语法

```
mail(to,subject,message,headers,parameters)
```

参数	描述
to	必需。规定 email 接收者。
subject	必需。规定 email 的主题。注释：该参数不能包含任何新行字符。
message	必需。定义要发送的消息。应使用 LF (\n) 来分隔各行。
headers	可选。规定附加的标题，比如 From、Cc 以及 Bcc。 应当使用 CRLF (\r\n) 分隔附加的标题。
parameters	可选。对邮件发送程序规定额外的参数。

注释：PHP 需要一个已安装且正在运行的邮件系统，以便使邮件函数可用。所用的程序通过在 php.ini 文件中的配置设置进行定义。请在我们的 PHP Mail 参考手册阅读更多内容。

PHP 简易 E-Mail

通过 PHP 发送电子邮件的最简单的方式是发送一封文本 email。

在下面的例子中，我们首先声明变量($to, $subject, $message, $from, $headers)，然后我们在 mail() 函数中使用这些变量来发送了一封 e-mail：

```
<?php

$to = "someone@example.com";
$subject = "Test mail";
$message = "Hello! This is a simple email message.";
$from = "someonelse@example.com";
$headers = "From: $from";
mail($to,$subject,$message,$headers);
echo "Mail Sent.";

?>
```

PHP Mail Form

通过 PHP，您能够在自己的站点制作一个反馈表单。下面的例子向指定的 e-mail 地址发送了一条文本消息：

```
<html>
<body>

<?php
if (isset($_REQUEST['email']))
//if "email" is filled out, send email
  {
  //send email
  $email = $_REQUEST['email'] ;
  $subject = $_REQUEST['subject'] ;
  $message = $_REQUEST['message'] ;
  mail( "someone@example.com", "Subject: $subject",
  $message, "From: $email" );
  echo "Thank you for using our mail form";
  }
else
//if "email" is not filled out, display the form
  {
  echo "<form method='post' action='mailform.php'>
  Email: <input name='email' type='text' /><br />
  Subject: <input name='subject' type='text' /><br />
  Message:<br />
  <textarea name='message' rows='15' cols='40'>
  </textarea><br />
  <input type='submit' />
  </form>";
  }
?>

</body>
</html>
```

例子解释：

1. 首先，检查是否填写了邮件输入框
2. 如果未填写（比如在页面被首次访问时），输出 HTML 表单
3. 如果已填写（在表单被填写后），从表单发送邮件
4. 当点击提交按钮后，重新载入页面，显示邮件发送成功的消息

PHP Mail 参考手册

如需更多有关 PHP mail() 函数的信息，请访问我们的 PHP Mail 参考手册。

PHP 安全的电子邮件

在上一节中的 **PHP e-mail** 脚本中，存在着一个漏洞。

PHP E-mail 注入

首先，请看上一节中的 PHP 代码：

```
<html>
<body>

<?php
if (isset($_REQUEST['email']))
//if "email" is filled out, send email
  {
  //send email
  $email = $_REQUEST['email'] ;
  $subject = $_REQUEST['subject'] ;
  $message = $_REQUEST['message'] ;
  mail("someone@example.com", "Subject: $subject",
  $message, "From: $email" );
  echo "Thank you for using our mail form";
  }
else
//if "email" is not filled out, display the form
  {
  echo "<form method='post' action='mailform.php'>
  Email: <input name='email' type='text' /><br />
  Subject: <input name='subject' type='text' /><br />
  Message:<br />
  <textarea name='message' rows='15' cols='40'>
  </textarea><br />
  <input type='submit' />
  </form>";
  }
?>

</body>
</html>
```

以上代码存在的问题是，未经授权的用户可通过输入表单在邮件头部插入数据。

假如用户在表单中的输入框内加入这些文本，会出现什么情况呢？

```
someone@example.com%0ACc:person2@example.com
%0ABcc:person3@example.com,person3@example.com,
anotherperson4@example.com,person5@example.com
%0ABTo:person6@example.com
```

与往常一样，mail() 函数把上面的文本放入邮件头部，那么现在头部有了额外的 Cc:, Bcc: 以及 To: 字段。当用户点击提交按钮时，这封 e-mail 会被发送到上面所有的地址！

PHP 防止 E-mail 注入

防止 e-mail 注入的最好方法是对输入进行验证。

下面的代码与上一节类似，不过我们已经增加了检测表单中 email 字段的输入验证程序：

```
<html>
<body>
<?php
function spamcheck($field)
  {
  //filter_var() sanitizes the e-mail
  //address using FILTER_SANITIZE_EMAIL
  $field=filter_var($field, FILTER_SANITIZE_EMAIL);

  //filter_var() validates the e-mail
  //address using FILTER_VALIDATE_EMAIL
  if(filter_var($field, FILTER_VALIDATE_EMAIL))
    {
    return TRUE;
    }
  else
    {
    return FALSE;
    }
  }

if (isset($_REQUEST['email']))
  {//if "email" is filled out, proceed

  //check if the email address is invalid
  $mailcheck = spamcheck($_REQUEST['email']);
  if ($mailcheck==FALSE)
    {
    echo "Invalid input";
    }
  else
    {//send email
    $email = $_REQUEST['email'] ;
```

```
    $subject = $_REQUEST['subject'] ;
    $message = $_REQUEST['message'] ;
    mail("someone@example.com", "Subject: $subject",
    $message, "From: $email" );
    echo "Thank you for using our mail form";
    }
  }
else
  {//if "email" is not filled out, display the form
  echo "<form method='post' action='mailform.php'>
  Email: <input name='email' type='text' /><br />
  Subject: <input name='subject' type='text' /><br />
  Message:<br />
  <textarea name='message' rows='15' cols='40'>
  </textarea><br />
  <input type='submit' />
  </form>";
  }
?>

</body>
</html>
```

在上面的代码中，我们使用了 PHP 过滤器来对输入进行验证：

- FILTER_SANITIZE_EMAIL 从字符串中删除电子邮件的非法字符
- FILTER_VALIDATE_EMAIL 验证电子邮件地址

您可以在我们的 PHP 过滤器这一节中阅读更多有关过滤器的内容。

PHP 错误处理

在 **PHP** 中，默认的错误处理很简单。一条消息会被发送到浏览器，这条消息带有文件名、行号以及一条描述错误的消息。

PHP 错误处理

在创建脚本和 web 应用程序时，错误处理是一个重要的部分。如果您的代码缺少错误检测编码，那么程序看上去很不专业，也为安全风险敞开了大门。

本教程介绍了 PHP 中一些最为重要的错误检测方法。

我们将为您讲解不同的错误处理方法：

- 简单的 "die()" 语句
- 自定义错误和错误触发器
- 错误报告

基本的错误处理：使用 **die()** 函数

第一个例子展示了一个打开文本文件的简单脚本：

```
<?php
$file=fopen("welcome.txt","r");
?>
```

如果文件不存在，您会获得类似这样的错误：

```
**Warning**: fopen(welcome.txt) [function.fopen]: failed to open st
No such file or directory in **C:\webfolder\test.php** on line **2*
```

为了避免用户获得类似上面的错误消息，我们在访问文件之前检测该文件是否存在：

```
<?php
if(!file_exists("welcome.txt"))
 {
 die("File not found");
 }
else
 {
 $file=fopen("welcome.txt","r");
 }
?>
```

现在，假如文件不存在，您会得到类似这样的错误消息：

```
File not found
```

比起之前的代码，上面的代码更有效，这是由于它采用了一个简单的错误处理机制在错误之后终止了脚本。

不过，简单地终止脚本并不总是恰当的方式。让我们研究一下用于处理错误的备选的 PHP 函数。

创建自定义错误处理器

创建一个自定义的错误处理器非常简单。我们很简单地创建了一个专用函数，可以在 PHP 中发生错误时调用该函数。

该函数必须有能力处理至少两个参数 (error level 和 error message)，但是可以接受最多五个参数（可选的：file, line-number 以及 error context）：

语法

```
error_function(error_level,error_message,
error_file,error_line,error_context)
```

参数	描述
error_level	必需。为用户定义的错误规定错误报告级别。必须是一个值数。 参见下面的表格：错误报告级别。
error_message	必需。为用户定义的错误规定错误消息。
error_file	可选。规定错误在其中发生的文件名。
error_line	可选。规定错误发生的行号。
error_context	可选。规定一个数组，包含了当错误发生时在用的每个变量以及它们的值。

错误报告级别

这些错误报告级别是错误处理程序旨在处理的错误的不同的类型：

值	常量	描述
2	E_WARNING	非致命的 run-time 错误。不暂停脚本执行。
8	E_NOTICE	Run-time 通知。 脚本发现可能有错误发生，但也可能在脚本正常运行时发生。
256	E_USER_ERROR	致命的用户生成的错误。这类似于程序员使用 PHP 函数 trigger_error() 设置的 E_ERROR。
512	E_USER_WARNING	非致命的用户生成的警告。这类似于程序员使用 PHP 函数 trigger_error() 设置的 E_WARNING。
1024	E_USER_NOTICE	用户生成的通知。这类似于程序员使用 PHP 函数 trigger_error() 设置的 E_NOTICE。
4096	E_RECOVERABLE_ERROR	可捕获的致命错误。 类似 E_ERROR，但可被用户定义的处理程序捕获。(参见 set_error_handler())
8191	E_ALL	所有错误和警告，除级别 E_STRICT 以外。 （在 PHP 6.0，E_STRICT 是 E_ALL 的一部分）

现在，让我们创建一个处理错误的函数：

```
function customError($errno, $errstr)
 {
 echo "<b>Error:</b> [$errno] $errstr<br />";
 echo "Ending Script";
 die();
 }
```

上面的代码是一个简单的错误处理函数。当它被触发时，它会取得错误级别和错误消息。然后它会输出错误级别和消息，并终止脚本。

现在，我们已经创建了一个错误处理函数，我们需要确定在何时触发该函数。

Set Error Handler

PHP 的默认错误处理程序是内建的错误处理程序。我们打算把上面的函数改造为脚本运行期间的默认错误处理程序。

可以修改错误处理程序，使其仅应用到某些错误，这样脚本就可以不同的方式来处理不同的错误。不过，在本例中，我们打算针对所有错误来使用我们的自定义错误处理程序：

```
set_error_handler("customError");
```

由于我们希望我们的自定义函数来处理所有错误，set_error_handler() 仅需要一个参数，可以添加第二个参数来规定错误级别。

实例

通过尝试输出不存在的变量，来测试这个错误处理程序：

```
<?php
//error handler function
function customError($errno, $errstr)
 {
 echo "<b>Error:</b> [$errno] $errstr";
 }

//set error handler
set_error_handler("customError");

//trigger error
echo($test);
?>
```

以上代码的输出应该类似这样：

```
Error: [8] Undefined variable: test
```

触发错误

在脚本中用户输入数据的位置，当用户的输入无效时触发错误的很有用的。在 PHP 中，这个任务由 trigger_error() 完成。

例子

在本例中，如果 "test" 变量大于 "1"，就会发生错误：

```
<?php
$test=2;
if ($test>1)
{
trigger_error("Value must be 1 or below");
}
?>
```

以上代码的输出应该类似这样：

```
Notice: Value must be 1 or below
in C:\webfolder\test.php on line 6
```

您可以在脚本中任何位置触发错误，通过添加的第二个参数，您能够规定所触发的错误级别。

可能的错误类型：

- E_USER_ERROR - 致命的用户生成的 run-time 错误。错误无法恢复。脚本执行被中断。
- E_USER_WARNING - 非致命的用户生成的 run-time 警告。脚本执行不被中断。
- E_USER_NOTICE - 默认。用户生成的 run-time 通知。脚本发现了可能的错误，也有可能在脚本运行正常时发生。

例子

在本例中，如果 "test" 变量大于 "1"，则发生 E_USER_WARNING 错误。如果发生了 E_USER_WARNING，我们将使用我们的自定义错误处理程序并结束脚本：

```
<?php
//error handler function
function customError($errno, $errstr)
 {
 echo "<b>Error:</b> [$errno] $errstr<br />";
 echo "Ending Script";
 die();
 }

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1)
 {
 trigger_error("Value must be 1 or below",E_USER_WARNING);
 }
?>
```

以上代码的输出应该类似这样：

```
Error: [512] Value must be 1 or below
Ending Script
```

现在，我们已经学习了如何创建自己的 error，以及如何触发它们，现在我们研究一下错误记录。

错误记录

默认地，根据在 php.ini 中的 error_log 配置，PHP 向服务器的错误记录系统或文件发送错误记录。通过使用 error_log() 函数，您可以向指定的文件或远程目的地发送错误记录。

通过电子邮件向您自己发送错误消息，是一种获得指定错误的通知的好办法。

通过 E-Mail 发送错误消息

在下面的例子中，如果特定的错误发生，我们将发送带有错误消息的电子邮件，并结束脚本：

```
<?php
//error handler function
function customError($errno, $errstr)
 {
 echo "<b>Error:</b> [$errno] $errstr<br />";
 echo "Webmaster has been notified";
 error_log("Error: [$errno] $errstr",1,
 "someone@example.com","From: webmaster@example.com");
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>1)
 {
 trigger_error("Value must be 1 or below",E_USER_WARNING);
 }
?>
```

以上代码的输出应该类似这样：

```
Error: [512] Value must be 1 or below
Webmaster has been notified
```

接收自以上代码的邮件类似这样：

```
Error: [512] Value must be 1 or below
```

这个方法不适合所有的错误。常规错误应当通过使用默认的 PHP 记录系统在服务器上进行记录。

PHP 异常处理

异常（**Exception**）用于在指定的错误发生时改变脚本的正常流程。

什么是异常？

PHP 5 提供了一种新的面向对象的错误处理方法。

异常处理用于在指定的错误（异常）情况发生时改变脚本的正常流程。这种情况称为异常。

当异常被触发时，通常会发生：

- 当前代码状态被保存
- 代码执行被切换到预定义的异常处理器函数
- 根据情况，处理器也许会从保存的代码状态重新开始执行代码，终止脚本执行，或从代码中另外的位置继续执行脚本

我们将展示不同的错误处理方法：

- 异常的基本使用
- 创建自定义的异常处理器
- 多个异常
- 重新抛出异常
- 设置顶层异常处理器

异常的基本使用

当异常被抛出时，其后的代码不会继续执行，PHP 会尝试查找匹配的 "catch" 代码块。

如果异常没有被捕获，而且又没用使用 set_exception_handler() 作相应的处理的话，那么将发生一个严重的错误（致命错误），并且输出 "Uncaught Exception"（未捕获异常）的错误消息。

让我们尝试抛出一个异常，同时不去捕获它：

```
<?php
//create function with an exception
function checkNum($number)
 {
 if($number>1)
  {
  throw new Exception("Value must be 1 or below");
  }
 return true;
 }

//trigger exception
checkNum(2);
?>
```

上面的代码会获得类似这样的一个错误：

```
Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6
```

Try, throw 和 catch

要避免上面例子出现的错误，我们需要创建适当的代码来处理异常。

正确的处理程序应当包括：

1. Try - 使用异常的函数应该位于 "try" 代码块内。如果没有触发异常，则代码将照常继续执行。但是如果异常被触发，会抛出一个异常。
2. Throw - 这里规定如何触发异常。每一个 "throw" 必须对应至少一个 "catch"
3. Catch - "catch" 代码块会捕获异常，并创建一个包含异常信息的对象

让我们触发一个异常：

```
<?php
//创建可抛出一个异常的函数
function checkNum($number)
 {
 if($number>1)
  {
  throw new Exception("Value must be 1 or below");
  }
 return true;
 }

//在 "try" 代码块中触发异常
try
 {
 checkNum(2);
 //If the exception is thrown, this text will not be shown
 echo 'If you see this, the number is 1 or below';
 }

//捕获异常
catch(Exception $e)
 {
 echo 'Message: ' .$e->getMessage();
 }
?>
```

上面代码将获得类似这样一个错误：

```
Message: Value must be 1 or below
```

例子解释：

上面的代码抛出了一个异常，并捕获了它：

1. 创建 checkNum() 函数。它检测数字是否大于 1。如果是，则抛出一个异常。
2. 在 "try" 代码块中调用 checkNum() 函数。
3. checkNum() 函数中的异常被抛出
4. "catch" 代码块接收到该异常，并创建一个包含异常信息的对象 ($e)。
5. 通过从这个 exception 对象调用 $e->getMessage()，输出来自该异常的错误消息

不过，为了遵循“每个 throw 必须对应一个 catch”的原则，可以设置一个顶层的异常处理器来处理漏掉的错误。

创建一个自定义的 Exception 类

创建自定义的异常处理程序非常简单。我们简单地创建了一个专门的类，当 PHP 中发生异常时，可调用其函数。该类必须是 exception 类的一个扩展。

这个自定义的 exception 类继承了 PHP 的 exception 类的所有属性，您可向其添加自定义的函数。

我们开始创建 exception 类：

```
<?php
class customException extends Exception
 {
 public function errorMessage()
  {
  //error message
  $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFi
  .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address
  return $errorMsg;
  }
 }

$email = "someone@example...com";

try
 {
 //check if
 if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
  {
  //throw exception if email is not valid
  throw new customException($email);
  }
 }

catch (customException $e)
 {
 //display custom message
 echo $e->errorMessage();
 }
?>
```

这个新的类是旧的 exception 类的副本，外加 errorMessage() 函数。正因为它是旧类的副本，因此它从旧类继承了属性和方法，我们可以使用 exception 类的方法，比如 getLine() 、 getFile() 以及 getMessage()。

例子解释：

上面的代码抛出了一个异常，并通过一个自定义的 exception 类来捕获它：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧类的所有属性和方法。

2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一条错误消息
3. 把 $email 变量设置为不合法的 e-mail 地址字符串
4. 执行 "try" 代码块，由于 e-mail 地址不合法，因此抛出一个异常
5. "catch" 代码块捕获异常，并显示错误消息

多个异常

可以为一段脚本使用多个异常，来检测多种情况。

可以使用多个 if..else 代码块，或一个 switch 代码块，或者嵌套多个异常。这些异常能够使用不同的 exception 类，并返回不同的错误消息：

```
<?php
class customException extends Exception
{
public function errorMessage()
{
//error message
$errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile
.': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
return $errorMsg;
}
}

$email = "someone@example.com";

try
 {
 //check if
 if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
  {
  //throw exception if email is not valid
  throw new customException($email);
  }
 //check for "example" in mail address
 if(strpos($email, "example") !== FALSE)
  {
  throw new Exception("$email is an example e-mail");
  }
 }

catch (customException $e)
 {
 echo $e->errorMessage();
 }

catch(Exception $e)
 {
 echo $e->getMessage();
 }
?>
```

例子解释：

上面的代码测试了两种条件，如何任何条件不成立，则抛出一个异常：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧类的所有属性和方法。
2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一个错误消息。
3. 执行 "try" 代码块，在第一个条件下，不会抛出异常。

4. 由于 e-mail 含有字符串 "example"，第二个条件会触发异常。
5. "catch" 代码块会捕获异常，并显示恰当的错误消息

如果没有捕获 customException，紧紧捕获了 base exception，则在那里处理异常。

重新抛出异常

有时，当异常被抛出时，您也许希望以不同于标准的方式对它进行处理。可以在一个 "catch" 代码块中再次抛出异常。

脚本应该对用户隐藏系统错误。对程序员来说，系统错误也许很重要，但是用户对它们并不感兴趣。为了让用户更容易使用，您可以再次抛出带有对用户比较友好的消息的异常：

```
<?php
class customException extends Exception
 {
 public function errorMessage()
  {
  //error message
  $errorMsg = $this->getMessage().' is not a valid E-Mail address.
  return $errorMsg;
  }
 }

$email = "someone@example.com";

try
 {
 try
  {
  //check for "example" in mail address
  if(strpos($email, "example") !== FALSE)
   {
   //throw exception if email is not valid
   throw new Exception($email);
   }
  }
 catch(Exception $e)
  {
  //re-throw exception
  throw new customException($email);
  }
 }

catch (customException $e)
 {
 //display custom message
 echo $e->errorMessage();
 }
?>
```

例子解释：

上面的代码检测在邮件地址中是否含有字符串 "example"。如果有，则再次抛出异常：

1. customException() 类是作为旧的 exception 类的一个扩展来创建的。这样它就继承了旧类的所有属性和方法。
2. 创建 errorMessage() 函数。如果 e-mail 地址不合法，则该函数返回一个错误消息。
3. 把 $email 变量设置为一个有效的邮件地址，但含有字符串 "example"。
4. "try" 代码块包含另一个 "try" 代码块，这样就可以再次抛出异常。

5. 由于 e-mail 包含字符串 "example"，因此触发异常。
6. "catch" 捕获到该异常，并重新抛出 "customException"。
7. 捕获到 "customException"，并显示一条错误消息。

如果在其目前的 "try" 代码块中异常没有被捕获，则它将在更高层级上查找 catch 代码块。

设置顶层异常处理器 （Top Level Exception Handler）

set_exception_handler() 函数可设置处理所有未捕获异常的用户定义函数。

```
<?php
function myException($exception)
{
echo "<b>Exception:</b> " , $exception->getMessage();
}

set_exception_handler('myException');

throw new Exception('Uncaught Exception occurred');
?>
```

以上代码的输出应该类似这样：

```
Exception: Uncaught Exception occurred
```

在上面的代码中，不存在 "catch" 代码块，而是触发顶层的异常处理程序。应该使用此函数来捕获所有未被捕获的异常。

异常的规则

- 需要进行异常处理的代码应该放入 try 代码块内，以便捕获潜在的异常。
- 每个 try 或 throw 代码块必须至少拥有一个对应的 catch 代码块。
- 使用多个 catch 代码块可以捕获不同种类的异常。
- 可以在 try 代码块内的 catch 代码块中再次抛出（re-thrown）异常。

简而言之：如果抛出了异常，就必须捕获它。

PHP 过滤器（Filter）

- MySQL 简介

PHP 过滤器用于验证和过滤来自非安全来源的数据，比如用户的输入。

什么是 PHP 过滤器？

PHP 过滤器用于验证和过滤来自非安全来源的数据。

验证和过滤用户输入或自定义数据是任何 Web 应用程序的重要组成部分。

设计 PHP 的过滤器扩展的目的是使数据过滤更轻松快捷。

为什么使用过滤器？

几乎所有 web 应用程序都依赖外部的输入。这些数据通常来自用户或其他应用程序（比如 web 服务）。通过使用过滤器，您能够确保应有程序获得正确的输入类型。

您应该始终对外部数据进行过滤！

输入过滤是最重要的应用程序安全课题之一。

什么是外部数据？

- 来自表单的输入数据
- Cookies
- 服务器变量
- 数据库查询结果

函数和过滤器

如需过滤变量，请使用下面的过滤器函数之一：

- filter_var() - 通过一个指定的过滤器来过滤单一的变量
- filter_var_array() - 通过相同的或不同的过滤器来过滤多个变量
- filter_input - 获取一个输入变量，并对它进行过滤
- filter_input_array - 获取多个输入变量，并通过相同的或不同的过滤器对它们进行过滤

在下面的例子中，我们用 filter_var() 函数验证了一个整数：

```
<?php
$int = 123;

if(!filter_var($int, FILTER_VALIDATE_INT))
 {
 echo("Integer is not valid");
 }
else
 {
 echo("Integer is valid");
 }
?>
```

上面的代码使用了 "FILTER_VALIDATE_INT" 过滤器来过滤变量。由于这个整数是合法的，因此代码的输出是："Integer is valid"。

假如我们尝试使用一个非整数的变量，则输出是："Integer is not valid"。

如需完整的函数和过滤器列表，请访问我们的 PHP Filter 参考手册。

Validating 和 Sanitizing

有两种过滤器：

Validating 过滤器：

- 用于验证用户输入
- 严格的格式规则（比如 URL 或 E-Mail 验证）
- 如果成功则返回预期的类型，如果失败则返回 FALSE

Sanitizing 过滤器：

- 用于允许或禁止字符串中指定的字符
- 无数据格式规则
- 始终返回字符串

选项和标志

选项和标志用于向指定的过滤器添加额外的过滤选项。

不同的过滤器有不同的选项和标志。

在下面的例子中，我们用 filter_var() 和 "min_range" 以及 "max_range" 选项验证了一个整数：

```
<?php
$var=300;
 $int_options = array(
"options"=>array
 (
 "min_range"=>0,
 "max_range"=>256
 )
);
if(!filter_var($var, FILTER_VALIDATE_INT, $int_options))
 {
 echo("Integer is not valid");
 }
else
 {
 echo("Integer is valid");
 }
?>
```

就像上面的代码一样，选项必须放入一个名为 "options" 的相关数组中。如果使用标志，则不需在数组内。

由于整数是 "300"，它不在指定的范围内，以上代码的输出将是 "Integer is not valid"。

如需完整的函数及过滤器列表，请访问 W3School 提供的 PHP Filter 参考手册。您可以看到每个过滤器的可用选项和标志。

验证输入

让我们试着验证来自表单的输入。

我们需要作的第一件事情是确认是否存在我们正在查找的输入数据。

然后我们用 filter_input() 函数过滤输入的数据。

在下面的例子中，输入变量 "email" 被传到 PHP 页面：

```
<?php
if(!filter_has_var(INPUT_GET, "email"))
 {
 echo("Input type does not exist");
 }
else
 {
 if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL))
  {
  echo "E-Mail is not valid";
  }
 else
  {
  echo "E-Mail is valid";
  }
 }
?>
```

例子解释：

上面的例子有一个通过 "GET" 方法传送的输入变量 (email)：

1. 检测是否存在 "GET" 类型的 "email" 输入变量
2. 如果存在输入变量，检测它是否是有效的邮件地址

净化输入

让我们试着清理一下从表单传来的 URL。

首先，我们要确认是否存在我们正在查找的输入数据。

然后，我们用 filter_input() 函数来净化输入数据。

在下面的例子中，输入变量 "url" 被传到 PHP 页面：

```
<?php
if(!filter_has_var(INPUT_POST, "url"))
 {
 echo("Input type does not exist");
 }
else
 {
 $url = filter_input(INPUT_POST, "url", FILTER_SANITIZE_URL);
 }
?>
```

例子解释：

上面的例子有一个通过 "POST" 方法传送的输入变量 (url)：

1. 检测是否存在 "POST" 类型的 "url" 输入变量
2. 如果存在此输入变量，对其进行净化（删除非法字符），并将其存储在 $url 变量中

假如输入变量类似这样："http://www.W3非o法ol.com.c字符n/"，则净化后的 $url 变量应该是这样的：

```
http://www.W3School.com.cn/
```

过滤多个输入

表单通常由多个输入字段组成。为了避免对 filter_var 或 filter_input 重复调用，我们可以使用 filter_var_array 或 the filter_input_array 函数。

在本例中，我们使用 filter_input_array() 函数来过滤三个 GET 变量。接收到的 GET 变量是一个名字、一个年龄以及一个邮件地址：

```
<?php
$filters = array
 (
 "name" => array
  (
  "filter"=>FILTER_SANITIZE_STRING
  ),
 "age" => array
  (
  "filter"=>FILTER_VALIDATE_INT,
  "options"=>array
   (
   "min_range"=>1,
   "max_range"=>120
   )
  ),
 "email"=> FILTER_VALIDATE_EMAIL,
 );

$result = filter_input_array(INPUT_GET, $filters);

if (!$result["age"])
 {
 echo("Age must be a number between 1 and 120.<br />");
 }
elseif(!$result["email"])
 {
 echo("E-Mail is not valid.<br />");
 }
else
 {
 echo("User input is valid");
 }
?>
```

例子解释：

上面的例子有三个通过 "GET" 方法传送的输入变量 (name, age and email)

1. 设置一个数组，其中包含了输入变量的名称，以及用于指定的输入变量的过滤器
2. 调用 filter_input_array 函数，参数包括 GET 输入变量及刚才设置的数组
3. 检测 $result 变量中的 "age" 和 "email" 变量是否有非法的输入。（如果存在非法输入，）

filter_input_array() 函数的第二个参数可以是数组或单一过滤器的 ID。

如果该参数是单一过滤器的 ID，那么这个指定的过滤器会过滤输入数组中所有的值。

如果该参数是一个数组，那么此数组必须遵循下面的规则：

- 必须是一个关联数组，其中包含的输入变量是数组的键（比如 "age" 输入变量）
- 此数组的值必须是过滤器的 ID ，或者是规定了过滤器、标志以及选项的数组

使用 Filter Callback

通过使用 FILTER_CALLBACK 过滤器，可以调用自定义的函数，把它作为一个过滤器来使用。这样，我们就拥有了数据过滤的完全控制权。

您可以创建自己的自定义函数，也可以使用已有的 PHP 函数。

规定您准备用到过滤器函数的方法，与规定选项的方法相同。

在下面的例子中，我们使用了一个自定义的函数把所有 "_" 转换为空格：

```
<?php
function convertSpace($string)
{
return str_replace("_", " ", $string);
}

$string = "Peter_is_a_great_guy!";

echo filter_var($string, FILTER_CALLBACK, array("options"=>"convert
?>
```

以上代码的结果是这样的：

```
Peter is a great guy!
```

例子解释：

上面的例子把所有 "_" 转换成空格：

1. 创建一个把 "_" 替换为空格的函数
2. 调用 filter_var() 函数，它的参数是 FILTER_CALLBACK 过滤器以及包含我们的函数的数组

- MySQL 简介

PHP JSON

本章节我们将为大家介绍如何使用 PHP 语言来编码和解码 JSON 对象。

环境配置

在 php5.2.0 及以上版本已经内置 JSON 扩展。

JSON 函数

函数	描述
json_encode	对变量进行 JSON 编码
json_decode	对 JSON 格式的字符串进行解码，转换为 PHP 变量
json_last_error	返回最后发生的错误

json_encode

PHP json_encode() 用于对变量进行 JSON 编码，该函数如果执行成功返回 JSON 数据，否则返回 FALSE 。

语法

```
string json_encode ( $value [, $options = 0 ] )
```

参数

- **value**: 要编码的值。该函数只对 UTF-8 编码的数据有效。
- **options**:由以下常量组成的二进制掩码：JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS, JSON_NUMERIC_CHECK,JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT

实例

以下实例演示了如何将 PHP 数组转换为 JSON 格式数据：

```
<?php
   $arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
   echo json_encode($arr);
?>
```

以上代码执行结果为：

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

以下实例演示了如何将 PHP 对象转换为 JSON 格式数据：

```
?php
   class Emp {
       public $name = "";
       public $hobbies  = "";
       public $birthdate = "";
   }
   $e = new Emp();
   $e->name = "sachin";
   $e->hobbies  = "sports";
   $e->birthdate = date('m/d/Y h:i:s a', "8/5/1974 12:20:03 p");
   $e->birthdate = date('m/d/Y h:i:s a', strtotime("8/5/1974 12:20

   echo json_encode($e);
?>
```

以上代码执行结果为：

```
{"name":"sachin","hobbies":"sports","birthdate":"08\/05\/1974 12:2(
```

json_decode

PHP json_decode() 函数用于对 JSON 格式的字符串进行解码，并转换为 PHP 变量。

语法

```
mixed json_decode ($json [,$assoc = false [, $depth = 512 [, $optic
```

参数

- **json_string**: 待解码的 JSON 字符串，必须是 UTF-8 编码数据
- **assoc**: 当该参数为 TRUE 时，将返回数组，FALSE 时返回对象。
- **depth**: 整数类型的参数，它指定递归深度
- **options**: 二进制掩码，目前只支持 JSON_BIGINT_AS_STRING 。

实例

以下实例演示了如何解码 JSON 数据：

```
<?php
   $json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

   var_dump(json_decode($json));
   var_dump(json_decode($json, true));
?>
```

以上代码执行结果为：

```
object(stdClass)#1 (5) {
    ["a"] => int(1)
    ["b"] => int(2)
    ["c"] => int(3)
    ["d"] => int(4)
    ["e"] => int(5)
}

array(5) {
    ["a"] => int(1)
    ["b"] => int(2)
    ["c"] => int(3)
    ["d"] => int(4)
    ["e"] => int(5)
}
```

PHP 数据库

PHP MySQL 简介

通过 PHP，您可以连接和操作数据库。

MySQL 是跟 PHP 配套使用的最流行的开源数据库系统。

如果想学习更多 MySQL 知识可以查看本站 MySQL 教程。

MySQL 是什么？

- MySQL 是一种在 Web 上使用的数据库系统。
- MySQL 是一种在服务器上运行的数据库系统。
- MySQL 不管在小型还是大型应用程序中，都是理想的选择。
- MySQL 是非常快速，可靠，且易于使用的。
- MySQL 支持标准的 SQL。
- MySQL 在一些平台上编译。
- MySQL 是免费下载使用的。
- MySQL 是由 Oracle 公司开发、发布和支持的。
- MySQL 是以公司创始人 Monty Widenius's daughter: My 命名的。

MySQL 中的数据存储在表中。表格是一个相关数据的集合，它包含了列和行。

在分类存储信息时，数据库非常有用。一个公司的数据库可能拥有以下表：

- Employees
- Products
- Customers
- Orders

PHP + MySQL

- PHP 与 MySQL 结合是跨平台的。（您可以在 Windows 上开发，在 Unix 平台上应用。）

查询

查询是一种询问或请求。

通过 MySQL，我们可以向数据库查询具体的信息，并得到返回的记录集。

请看下面的查询（使用标准 SQL）：

```
SELECT LastName FROM Employees
```

上面的查询选取了 "Employees" 表中 "LastName" 列的所有数据。

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

下载 MySQL 数据库

如果您的 PHP 服务器没有 MySQL 数据库，可以在此免费下载 MySQL：http://www.mysql.com。

关于 MySQL 数据库的事实

关于 MySQL 的一点很棒的特性是，可以对它进行缩减，来支持嵌入的数据库应用程序。也许正因为如此，许多人认为 MySQL 仅仅能处理中小型的系统。

事实上，对于那些支持巨大数据和访问量的网站（比如 Friendster、Yahoo、Google），MySQL 是事实上的标准数据库。

这个地址提供了使用 MySQL 的公司的概览：http://www.mysql.com/customers/。

PHP 连接 MySQL

PHP 5 及以上版本建议使用以下方式连接 MySQL :

- **MySQLi extension** ("i" 意为 improved)
- **PDO (PHP Data Objects)**

在 PHP 早起版本中我们使用 MySQL 扩展。但该扩展在 2012 年开始不建议使用。

我是该用 MySQLi ， 还是 PDO?

如果你需要一个简短的回答，即 "你习惯哪个就用哪个"。

MySQLi 和 PDO 有它们自己的优势 ：

PDO 应用在 12 种不同数据库中， MySQLi 只针对 MySQL 数据库。

所以，如果你的项目需要在多种数据库中切换，建议使用 PDO ，这样你只需要修改连接字符串和部门查询语句即可。 使用 MySQLi, 如果不同数据库，你需要重新所有代码，包括查询。

两者都是面向对象, 但 MySQLi 还提供了 API 接口。

两者都支持预处理语句。 预处理语句可以防止 SQL 注入，对于 web 项目的安全性是非常重要的。

MySQLi 和 PDO 连接 MySQL 实例

在本章节及接下来的章节中，我们会使用以下三种方式来演示 PHP 操作 MySQL:

- MySQLi (面向对象)
- MySQLi (面向过程)
- PDO

MySQLi Installation

Linux 和 Windows: 在 php5 mysql 包安装时 MySQLi 扩展多事情况下是自动安装的。

安装详细信息，请查看： http://php.net/manual/en/mysqli.installation.php

PDO 安装

For 安装详细信息，请查看： http://php.net/manual/en/pdo.installation.php

连接 MySQL

在我们访问 MySQL 数据库前，我们需要先连接到数据库服务器：

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// 创建连接
$conn = new mysqli($servername, $username, $password);

// 检测连接
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

 注意在以上面向对象的实例中 $connect_error 是在 PHP 5.2.9 和 5.3.0 中添加的。如果你需要兼容更早版本 请使用以下代码替换:

```
// 检测连接
if (mysqli_connect_error()) {
    die("Database connection failed: " . mysqli_connect_error());
}
```

实例 (MySQLi - 面向过程)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// 创建连接
$conn = mysqli_connect($servername, $username, $password);

// 检测连接
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username
    echo "Connected successfully";
    }
catch(PDOException $e)
    {
    echo $e->getMessage();
    }
?>
```

 注意在以上 PDO 实例中我们已经指定了数据库 (myDB)。PDO 在连接过程需要设置数据库名。如果没有指定，则会抛出异常。

关闭连接

连接在脚本执行完后会自动关闭。你也可以使用以下代码来关闭连接：

实例 (MySQLi - 面向对象)

```
$conn->close();
```

实例 (MySQLi - 面向过程)

```
mysqli_close($conn);
```

实例 (PDO)

```
$conn = null;
```

PHP MySQL 创建数据库

数据库存有一个或多个表。

你需要 CREATE 权限来创建或删除 MySQL 数据库。

使用 MySQLi 和 PDO 创建 MySQL 数据库

CREATE DATABASE 语句用于在 MySQL 中创建数据库。

在下面的实例中，创建了一个名为 "myDB" 的数据库：

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// 创建连接
$conn = new mysqli($servername, $username, $password);
// 检测连接
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . $conn->error;
}

$conn->close();
?>
```

 注意： 当你创建一个新的数据库时，你必须为 mysqli 对象指定三个参数 (servername, username 和 password)。

Tip: 如果你使用其他端口（默认为3306），为数据库参数添加空字符串，如: new mysqli("localhost", "username", "password", "", port)

实例 (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// 创建连接
$conn = mysqli_connect($servername, $username, $password);
// 检测连接
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully";
} else {
    echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

注意： 以下使用 PDO 实例创建数据库 "myDBPDO":

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username
    // 设置 PDO 错误模式为异常
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "CREATE DATABASE myDBPDO";
    // 使用 exec() ，因为没有结果返回
    $conn->exec($sql);
    echo "Database created successfully<br>";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

提示： 使用 PDO 的最大好处是在数据库查询过程出现问题时可以使用异常类来 处理问题。如果 try{ } 代码块出现异常，脚本会停止执行并会跳到第一个 catch(){ } 代码块执行代码。 在以上捕获的代码块中我们输出了 SQL 语句并生成错误信息。

PHP 创建 MySQL 表

一个数据表有一个唯一名称，并有行和列组成。

使用 MySQLi 和 PDO 创建 MySQL 表

CREATE TABLE 语句用于创建 MySQL 表。

我们将创建一个名为 "MyGuests" 的表，有 5 个列： "id", "firstname", "lastname", "email" 和 "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)
```

上表中的注意事项:

数据类型指定列可以存储什么类型的数据。完整的数据类型请参考我们的 数据类型参考手册。

在设置了数据类型后，你可以为没个列指定其他选项的属性：

- NOT NULL - 没一行都必须含有值（不能为空），null 值是不允许的。
- DEFAULT value - 设置默认值
- UNSIGNED - 使用无符号数值类型，0 及正数
- AUTO INCREMENT - 设置 MySQL 字段的值在新增记录时每次自动增长 1
- PRIMARY KEY - 设置数据表中每条记录的唯一标识。 通常列的 PRIMARY KEY 设置为 ID 数值，与 AUTO_INCREMENT 一起使用。

每个表都应该有一个主键(本列为 "id" 列)，主键必须包含唯一的值。

以下实例展示了如何在 PHP 中创建表：

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检测连接
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . $conn->error;
}

$conn->close();
?>
```

实例 (MySQLi - 面向过程)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = mysqli_connect($servername, $username, $password, $dbname);
// 检测连接
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
    echo "Table MyGuests created successfully";
} else {
    echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $usern
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // sql to create table
    $sql = "CREATE TABLE MyGuests (
    id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    email VARCHAR(50),
    reg_date TIMESTAMP
    )";

    // use exec() because no results are returned
    $conn->exec($sql);
    echo "Table MyGuests created successfully";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

PHP MySQL 插入数据

使用 MySQLi 和 PDO 向 MySQL 插入数据

在创建完数据库和表后，我们可以向表中添加数据。

以下为一些语法规则：

- PHP 中 SQL 查询语句必须使用引号
- 在 SQL 查询语句中的字符串值必须加引号
- 数值的值不需要引号
- NULL 值不需要引号

INSERT INTO 语句通常用于向 MySQL 表添加新的记录：

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

学习更多关于 SQL 知识，请查看我们的 SQL 教程。

在前面的几个章节中我们已经创建了表 "MyGuests"，表字段有: "id", "firstname", "lastname", "email" 和 "reg_date"。 现在，让我们开始向表填充数据。

>  注意： 如果列设置 AUTO_INCREMENT (如 "id" 列) 或 TIMESTAMP (如 "reg_date" 列),， 我们就不需要在 SQL 查询语句中指定值； MySQL 会自动为该列添加值。

以下实例向 "MyGuests" 表添加了新的记录:

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检测连接
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

实例 (MySQLi - 面向过程)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = mysqli_connect($servername, $username, $password, $dbname);
// 检测连接
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $userr
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $sql = "INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('John', 'Doe', 'john@example.com')";
    // use exec() because no results are returned
    $conn->exec($sql);
    echo "New record created successfully";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

PHP MySQL 插入多条数据

使用 MySQLi 和 PDO 向 MySQL 插入多条数据

mysqli_multi_query() 函数可用来执行多条SQL语句。

以下实例向 "MyGuests" 表添加了三条新的记录:

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建链接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检查链接
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if ($conn->multi_query($sql) === TRUE) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```


请注意，每个SQL语句必须用分号隔开。

实例 (MySQLi - 面向过程)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建链接
$conn = mysqli_connect($servername, $username, $password, $dbname);
// 检查链接
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if (mysqli_multi_query($conn, $sql)) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

实例 (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $usern
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // 开始事务
    $conn->beginTransaction();
    // SQL 语句
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('John', 'Doe', 'john@example.com')");
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('Mary', 'Moe', 'mary@example.com')");
    $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
    VALUES ('Julie', 'Dooley', 'julie@example.com')");

    // commit the transaction
    $conn->commit();
    echo "New records created successfully";
    }
catch(PDOException $e)
    {
    // roll back the transaction if something failed
    $conn->rollback();
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```

使用预处理语句

mysqli 扩展提供了第二种方式用于插入语句。

我们可以预处理语句及绑定参数。

mysql 扩展可以不带数据发送语句或查询到mysql数据库。 你可以向列关联或 "绑定" 变量。

Example (MySQLi 使用预处理语句)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
} else {
    $sql = "INSERT INTO MyGuests VALUES(?, ?, ?)";

    // 为 mysqli_stmt_prepare() 初始化 statement 对象
    $stmt = mysqli_stmt_init($conn);

    //预处理语句
    if (mysqli_stmt_prepare($stmt, $sql)) {
        // 绑定参数
        mysqli_stmt_bind_param($stmt, 'sss', $firstname, $lastname,

        // 设置参数并执行
        $firstname = 'John';
        $lastname = 'Doe';
        $email = 'john@example.com';
        mysqli_stmt_execute($stmt);

        $firstname = 'Mary';
        $lastname = 'Moe';
        $email = 'mary@example.com';
        mysqli_stmt_execute($stmt);

        $firstname = 'Julie';
        $lastname = 'Dooley';
        $email = 'julie@example.com';
        mysqli_stmt_execute($stmt);
    }
}
?>
```

我们可以看到以上实例中使用模块化来处理问题。我们可以通过创建代码块实现更简单的读取和管理。

注意参数的绑定。让我们看下 mysqli_stmt_bind_param() 中的代码：

```
mysqli_stmt_bind_param($stmt, 'sss', $firstname, $lastname, $email
```

该函数绑定参数查询并将参数传递给数据库。第二个参数是 "sss" 。以下列表展示了参数的类型。 s 字符告诉 mysql 参数是字符串。

This argument may be one of four types:

- i - integer
- d - double
- s - string
- b - BLOB

每个参数必须指定类型，来保证数据的安全性。通过类型的判断可以减少SQL注入漏洞带来的风险。

PHP MySQL 预处理语句

预处理语句对于防止 MySQL 注入是非常有用的。

预处理语句及绑定参数

预处理语句用于执行多个相同的 SQL 语句，并且执行效率更高。

预处理语句的工作原理如下：

1. 预处理：创建 SQL 语句模板并发送到数据库。预留的值使用参数 "?" 标记 。例如：INSERT INTO MyGuests VALUES(?, ?, ?)
2. 数据库解析，编译，对SQL语句模板执行查询优化，并存储结果不输出
3. 执行：最后，将应用绑定的值传递给参数（"?" 标记），数据库执行语句。应用可以多次执行语句，如果参数的值不一样。

相比于直接执行SQL语句，预处理语句有两个主要优点：

- 预处理语句大大减少了分析时间，只做了一次查询（虽然语句多次执行）
- 绑定参数减少了服务器带宽，你只需要发送查询的参数，而不是整个语句
- 预处理语句针对SQL注入是非常有用的，因为 参数值发送后使用不同的协议，保证了数据的合法性。

MySQLi 预处理语句

以下实例在 MySQLi 中使用了预处理语句，并绑定了相应的参数:

实例 (MySQLi 使用预处理语句)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);

// 检测连接
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests VALUES(?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// 设置参数并执行
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

解析以下实例的每行代码:

```
"INSERT INTO MyGuests VALUES(?, ?, ?)"
```

在 SQL 语句中，我们使用了问号 (?)，在此我们可以将问号替换为整型，字符串，双精度浮点型和布尔值。

接下来，让我们来看下 bind_param() 函数：

```
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

该函数绑定了 SQL 的参数，且告诉数据库参数的值。 "sss" 参数列处理其余参数的数据类型。s 字符告诉数据库该参数为字符串。

参数有以下四种类型:

- i - integer（整型）
- d - double（双精度浮点型）
- s - string（字符串）
- b - BLOB（布尔值）

每个参数都需要指定类型。

通过告诉数据库参数的数据类型，可以降低 SQL 注入的风险。

>  注意： 如果你想插入其他数据（用户输入），对数据的验证是非常重要的。

PDO 中的预处理语句

以下实例我们在 PDO 中使用了预处理语句并绑定参数:

实例 (PDO 使用预处理语句)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $usern
    // 设置 PDO 错误模式为异常
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // 预处理 SQL 并绑定参数
    $stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastna
    VALUES (:firstname, :lastname, :email)");
    $stmt->bindParam(':firstname', $firstname);
    $stmt->bindParam(':lastname', $lastname);
    $stmt->bindParam(':email', $email);

    // 插入行
    $firstname = "John";
    $lastname = "Doe";
    $email = "john@example.com";
    $stmt->execute();

    // 插入其他行
    $firstname = "Mary";
    $lastname = "Moe";
    $email = "mary@example.com";
    $stmt->execute();

    // 插入其他行
    $firstname = "Julie";
    $lastname = "Dooley";
    $email = "julie@example.com";
    $stmt->execute();

    echo "New records created successfully";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }
$conn = null;
?>
```

PHP MySQL 读取数据

从 MySQL 数据库读取数据

SELECT 语句用于从数据表中读取数据:

```
SELECT column_name(s) FROM table_name
```

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

以下实例中我们从表 MyGuests 读取了 id, firstname 和 lastname 列的数据并显示在页面上：

实例 (MySQLi - 面向对象)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// 创建连接
$conn = new mysqli($servername, $username, $password, $dbname);
// 检测连接
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
    // 输出每行数据
    while($row = $result->fetch_assoc()) {
        echo "<br> id: ". $row["id"]. " - Name: ". $row["firstname'
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

以下实例读取了 MyGuests 表的所有记录并显示在 HTML 表格中：

实例 (PDO)

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th><th>Email</

class TableRows extends RecursiveIteratorIterator {
    function __construct($it) {
        parent::__construct($it, self::LEAVES_ONLY);
    }

    function current() {
        return "<td style='width: 150px; border: 1px solid black;'>
    }

    function beginChildren() {
        echo "<tr>";
    }

    function endChildren() {
        echo "</tr>" . "\n";
    }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $usern
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT * FROM MyGuests");
    $stmt->execute();

    // 设置结果集为关联数组
    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);

    foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAl
        echo $v;
    }
    $dsn = null;
    }
catch(PDOException $e)
    {
    echo "Error: " . $e->getMessage();
    }
$conn = null;
echo "</table>";
?>
```

PHP MySQL Where 子句

WHERE 子句用于过滤记录。

WHERE 子句

WHERE 子句用于提取满足指定标准的的记录。

语法

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
```

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

为了让 PHP 执行上面的语句，我们必须使用 mysqli_query() 函数。该函数用于向 MySQL 连接发送查询或命令。

实例

下面的实例将从 "Persons" 表中选取所有 FirstName='Peter' 的行：

```
<?php
$con=mysqli_connect("example.com","peter","abc123","my_db");
// Check connection
if (mysqli_connect_errno())
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$result = mysqli_query($con,"SELECT * FROM Persons
WHERE FirstName='Peter'");

while($row = mysqli_fetch_array($result))
{
echo $row['FirstName'] . " " . $row['LastName'];
echo "<br>";
}
?>
```

以上代码将输出：

```
Peter Griffin
```

PHP MySQL Order By 关键词

ORDER BY 关键词用于对记录集中的数据进行排序。

ORDER BY 关键词

ORDER BY 关键词用于对记录集中的数据进行排序。

ORDER BY 关键词默认对记录进行升序排序。

如果你想降序排序，请使用 DESC 关键字。

语法

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC
```

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

实例

下面的实例选取 "Persons" 表中存储的所有数据，并根据 "Age" 列对结果进行排序：

```
<?php
$con=mysqli_connect("example.com","peter","abc123","my_db");
// Check connection
if (mysqli_connect_errno())
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$result = mysqli_query($con,"SELECT * FROM Persons ORDER BY age");

while($row = mysqli_fetch_array($result))
{
echo $row['FirstName'];
echo " " . $row['LastName'];
echo " " . $row['Age'];
echo "<br>";
}

mysqli_close($con);
?>
```

以上结果将输出：

```
Glenn Quagmire 33
Peter Griffin 35
```

根据两列进行排序

可以根据多个列进行排序。当按照多个列进行排序时，只有第一列的值相同时才使用第二列：

```
SELECT column_name(s)
FROM table_name
ORDER BY column1, column2
```

PHP MySQL Update

UPDATE 语句用于中修改数据库表中的数据。

更新数据库中的数据

UPDATE 语句用于更新数据库表中已存在的记录。

语法

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

注释：请注意 UPDATE 语法中的 WHERE 子句。WHERE 子句规定了哪些记录需要更新。如果您想省去 WHERE 子句，所有的记录都会被更新！

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

为了让 PHP 执行上面的语句，我们必须使用 mysqli_query() 函数。该函数用于向 MySQL 连接发送查询或命令。

实例

在本教程的前面章节中，我们创建了一个名为 "Persons" 的表，如下所示：

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

下面的例子更新 "Persons" 表的一些数据：

```
<?php
$con=mysqli_connect("example.com","peter","abc123","my_db");
// Check connection
if (mysqli_connect_errno())
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_query($con,"UPDATE Persons SET Age=36
WHERE FirstName='Peter' AND LastName='Griffin'");

mysqli_close($con);
?>
```

在这次更新后，"Persons" 表如下所示：

FirstName	LastName	Age
Peter	Griffin	36
Glenn	Quagmire	33

PHP MySQL Delete

DELETE 语句用于从数据库表中删除行。

删除数据库中的数据

DELETE FROM 语句用于从数据库表中删除记录。

语法

```
DELETE FROM table_name
WHERE some_column = some_value
```

注释：请注意 DELETE 语法中的 WHERE 子句。WHERE 子句规定了哪些记录需要删除。如果您想省去 WHERE 子句，所有的记录都会被删除！

如需学习更多关于 SQL 的知识，请访问我们的 SQL 教程。

为了让 PHP 执行上面的语句，我们必须使用 mysqli_query() 函数。该函数用于向 MySQL 连接发送查询或命令。

实例

请看下面的 "Persons" 表：

FirstName	LastName	Age
Peter	Griffin	35
Glenn	Quagmire	33

下面的实例删除 "Persons" 表中所有 LastName='Griffin' 的记录：

```
<?php
$con=mysqli_connect("example.com","peter","abc123","my_db");
// Check connection
if (mysqli_connect_errno())
{
echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

mysqli_query($con,"DELETE FROM Persons WHERE LastName='Griffin'");

mysqli_close($con);
?>
```

在这次删除后，"Persons" 表如下所示：

FirstName	LastName	Age
Glenn	Quagmire	33

PHP 数据库 ODBC

ODBC 是一种应用程序编程接口（Application Programming Interface，API），使我们有能力连接到某个数据源（比如一个 MS Access 数据库）。

创建 ODBC 连接

通过一个 ODBC 连接，您可以连接到您的网络中的任何计算机上的任何数据库，只要 ODBC 连接是可用的。

这是创建到达 MS Access 数据库的 ODBC 连接的方法：

1. 在控制面板中打开管理工具图标。
2. 双击其中的数据源**(ODBC)**图标。
3. 选择系统 **DSN** 选项卡。
4. 点击系统 DSN 选项卡中的添加。
5. 选择**Microsoft Access Driver**。点击完成。
6. 在下一个界面，点击选择来定位数据库。
7. 为数据库起一个数据源名**(DSN)**。
8. 点击确定。

请注意，必须在您的网站所在的计算机上完成这个配置。如果您的计算机上正在运行 Internet 信息服务(IIS)，上面的指令将会生效，但是如果您的网站位于远程服务器，您必须拥有对该服务器的物理访问权限，或者请您的主机提供商为您建立 DSN。

连接到 ODBC

odbc_connect() 函数用于连接到 ODBC 数据源。该函数有四个参数：数据源名、用户名、密码以及可选的指针类型。

odbc_exec() 函数用于执行 SQL 语句。

实例

下面的实例创建了到达名为 northwind 的 DSN 的连接，没有用户名和密码。然后创建并执行一条 SQL 语句：

```
$conn=odbc_connect('northwind','','');
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
```

取回记录

odbc_fetch_row() 函数用于从结果集中返回记录。如果能够返回行，则函数返回 true，否则返回 false。

该函数有两个参数：ODBC 结果标识符和可选的行号：

```
odbc_fetch_row($rs)
```

从记录中取回字段

odbc_result() 函数用于从记录中读取字段。该函数有两个参数：ODBC 结果标识符和字段编号或名称。

下面的代码行从记录中返回第一个字段的值：

```
$compname=odbc_result($rs,1);
```

下面的代码行返回名为 "CompanyName" 的字段的值：

```
$compname=odbc_result($rs,"CompanyName");
```

关闭 ODBC 连接

odbc_close() 函数用于关闭 ODBC 连接。

```
odbc_close($conn);
```

ODBC 实例

下面的实例展示了如何首先创建一个数据库连接，接着创建一个结果集，然后在 HTML 表格中显示数据。

```
<html>
<body>

<?php
$conn=odbc_connect('northwind','','');
if (!$conn)
{exit("Connection Failed: " . $conn);}
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
if (!$rs)
{exit("Error in SQL");}
echo "<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";
while (odbc_fetch_row($rs))
{
$compname=odbc_result($rs,"CompanyName");
$conname=odbc_result($rs,"ContactName");
echo "<tr><td>$compname</td>";
echo "<td>$conname</td></tr>";
}
odbc_close($conn);
echo "</table>";
?>

</body>
</html>
```

PHP XML

PHP XML Expat 解析器

内建的 **Expat** 解析器使在 **PHP** 中处理 **XML** 文档成为可能。

什么是 XML？

XML 用于描述数据，其焦点是数据是什么。XML 文件描述了数据的结构。

在 XML 中，没有预定义的标签。您必须定义自己的标签。

如果希望学习更多有关 XML 的内容，请访问我们的 XML 教程。

什么是 Expat？

如需读取和更新 - 创建创建并处理 - 一个 XML 文档，您需要 XML 解析器。

有两种基本的 XML 解析器类型：

- 基于树的解析器：这种解析器把 XML 文档转换为树型结构。它分析整篇文档，并提供了 API 来访问树种的元素，例如文档对象模型 (DOM)。
- 基于事件的解析器：将 XML 文档视为一系列的事件。当某个具体的事件发生时，解析器会调用函数来处理。

Expat 解析器是基于事件的解析器。

基于事件的解析器集中在 XML 文档的内容，而不是它们的结果。正因如此，基于事件的解析器能够比基于树的解析器更快地访问数据。

请看下面的 XML 片段：

```
<from>John</from>
```

基于事件的解析器把上面的 XML 报告为一连串的三个事件：

- 开始元素：from
- 开始 CDATA 部分，值：John
- 关闭元素： from

上面的 XML 范例包含了形式良好的 XML。不过这个例子是无效的 XML，因为没有与它关联的文档类型声明 (DTD)，也没有内嵌的 DTD。

不过，在使用 Expat 解析器时，这没有区别。Expat 是不检查有效性的解析器，忽略任何 DTD。

作为一款基于事件、非验证的 XML 解析器，Expat 快速且轻巧，十分适合 PHP 的 web 应用程序。

注释：XML 文档必须形式良好，否则 Expat 会生成错误。

安装

XML Expat 解析器是 PHP 核心的组成部分。无需安装就可以使用这些函数。

XML 文件

将在我们的例子中使用下面的 XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

初始化 XML 解析器

我们要在 PHP 中初始化 XML 解析器，为不同的 XML 事件定义处理器，然后解析这个 XML 文件。

例子

```
<?php

//Initialize the XML parser
$parser=xml_parser_create();

//Function to use at the start of an element
function start($parser,$element_name,$element_attrs)
  {
  switch($element_name)
    {
    case "NOTE":
    echo "-- Note --<br />";
    break;
    case "TO":
    echo "To: ";
    break;
    case "FROM":
    echo "From: ";
    break;
    case "HEADING":
```

```
    echo "Heading: ";
    break;
    case "BODY":
    echo "Message: ";
    }
  }

//Function to use at the end of an element
function stop($parser,$element_name)
  {
  echo "<br />";
  }

//Function to use when finding character data
function char($parser,$data)
  {
  echo $data;
  }

//Specify element handler
xml_set_element_handler($parser,"start","stop");

//Specify data handler
xml_set_character_data_handler($parser,"char");

//Open XML file
$fp=fopen("test.xml","r");

//Read data
while ($data=fread($fp,4096))
  {
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
  }

//Free the XML parser
xml_parser_free($parser);

?>
```

以上代码的输出：

```
-- Note --
To: George
From: John
Heading: Reminder
Message: Don't forget the meeting!
```

工作原理解释：

- 通过 xml_parser_create() 函数初始化 XML 解析器
- 创建配合不同事件处理程序的的函数
- 添加 xml_set_element_handler() 函数来定义，当解析器遇到开始和结束标签时执行哪个函数
- 添加 xml_set_character_data_handler() 函数来定义，当解析器遇到字符数据时执行哪个函数
- 通过 xml_parse() 函数来解析文件 "test.xml"
- 万一有错误的话，添加 xml_error_string() 函数把 XML 错误转换为文本说明
- 调用 xml_parser_free() 函数来释放分配给 xml_parser_create() 函数的内存

更多 **PHP Expat** 解析器的信息

如需更多有关 PHP Expat 函数的信息，请访问我们的 PHP XML Parser 参考手册。

PHP XML DOM

内建的 **DOM** 解析器使在 **PHP** 中处理 **XML** 文档成为可能。

什么是 DOM？

W3C DOM 提供了针对 HTML 和 XML 文档的标准对象集，以及用于访问和操作这些文档的标准接口。

W3C DOM 被分为不同的部分 (Core, XML 和 HTML) 和不同的级别 (DOM Level 1/2/3)：

- Core DOM - 为任何结构化文档定义标准的对象集
- XML DOM - 为 XML 文档定义标准的对象集
- HTML DOM - 为 HTML 文档定义标准的对象集

如果您希望学习更多有关 XML DOM 的知识，请访问我们的 XML DOM 教程。

XML 解析

如需读取和更新 - 创建创建并处理 - 一个 XML 文档，您需要 XML 解析器。

有两种基本的 XML 解析器类型：

- 基于树的解析器：这种解析器把 XML 文档转换为树型结构。它分析整篇文档，并提供了 API 来访问树种的元素，例如文档对象模型 (DOM)。
- 基于事件的解析器：将 XML 文档视为一系列的事件。当某个具体的事件发生时，解析器会调用函数来处理。

DOM 解析器是基于树的解析器。

请看下面的 XML 文档片段：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<from>John</from>
```

XML DOM 把 XML 视为一个树形结构：

- Level 1: XML 文档
- Level 2: 根元素: <from>
- Level 3: 文本元素: "John"

安装

DOM XML 解析器函数是 PHP 核心的组成部分。无需安装就可以使用这些函数。

XML 文件

将在我们的例子中使用下面的 XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

加载和输出 XML

我们需要初始化 XML 解析器，加载 XML，并把它输出：

例子

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

以上代码的输出：

```
George John Reminder Don't forget the meeting!
```

假如您在浏览器窗口中查看源代码，会看到下面这些 HTML：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

上面的例子创建了一个 DOMDocument-Object，并把 "note.xml" 中的 XML 载入这个文档对象中。

saveXML() 函数把内部 XML 文档放入一个字符串，这样我们就可以输出它。

循环 XML

我们要初始化 XML 解析器，加载 XML，并循环 <note> 元素的所有元素：

例子

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item)
  {
  print $item->nodeName . " = " . $item->nodeValue . "<br />";
  }
?>
```

以上代码的输出：

```
#text =
to = George
#text =
from = John
#text =
heading = Reminder
#text =
body = Don't forget the meeting!
#text =
```

在上面的例子中，您看到了每个元素之间存在空的文本节点。

当 XML 生成时，它通常会在节点之间包含空白。XML DOM 解析器把它们当作普通的元素，如果您不注意它们，有时会产生问题。

如果您希望学习更多有关 XML DOM 的知识，请访问我们的 XML DOM 教程。

PHP SimpleXML

SimpleXML 处理最普通的 **XML** 任务，其余的任务则交由其它扩展。

什么是 SimpleXML？

SimpleXML 是 PHP 5 中的新特性。在了解 XML 文档 layout 的情况下，它是一种取得元素属性和文本的便利途径。

与 DOM 或 Expat 解析器相比，SimpleXML 仅仅用几行代码就可以从元素中读取文本数据。

SimpleXML 可把 XML 文档转换为对象，比如：

- 元素 - 被转换为 SimpleXMLElement 对象的单一属性。当同一级别上存在多个元素时，它们会被置于数组中。
- 属性 - 通过使用关联数组进行访问，其中的下标对应属性名称。
- 元素数据 - 来自元素的文本数据被转换为字符串。如果一个元素拥有多个文本节点，则按照它们被找到的顺序进行排列。

当执行类似下列的基础任务时，SimpleXML 使用起来非常快捷：

- 读取 XML 文件
- 从 XML 字符串中提取数据
- 编辑文本节点或属性

不过，在处理高级 XML 时，比如命名空间，最好使用 Expat 解析器或 XML DOM。

安装

从 PHP 5.0 开始，SimpleXML 函数是 PHP 核心的组成部分。无需安装就可以使用这些函数。

使用 SimpleXML

下面是 XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

我们打算从上面的 XML 文件输出元素的名称和数据。

这是需要做的事情：

1. 加载 XML 文件
2. 取得第一个元素的名称
3. 使用 children() 函数创建在每个子节点上触发的循环
4. 输出每个子节点的元素名称和数据

例子

```
<?php
$xml = simplexml_load_file("test.xml");

echo $xml->getName() . "<br />";

foreach($xml->children() as $child)
  {
  echo $child->getName() . ": " . $child . "<br />";
  }
?>
```

以上代码的输出：

```
note
to: George
from: John
heading: Reminder
body: Don't forget the meeting!
```

更多有关 PHP SimpleXML 的信息

如需更多有关 PHP SimpleXML 的信息，请访问我们的 PHP SimpleXML 参考手册。

PHP AJAX

AJAX 简介

AJAX = Asynchronous JavaScript And XML（异步 JavaScript 及 XML）

AJAX 是 _A_synchronous _J_avaScript _A_nd _X_ML 的首字母缩写。

AJAX 并不是一种新的编程语言，而仅仅是一种新的技术，它可以创建更好、更快且交互性更强的 web 应用程序。

AJAX 使用 JavaScript 在 web 浏览器与 web 服务器之间来发送和接收数据。

通过在幕后与 web 服务器交换数据，而不是每当用户作出改变时重载整个 web 页面，AJAX 技术可以使网页更迅速地响应。

AJAX 基于开放的标准

AJAX 基于以下开放的标准：

- *JavaScript*
- *XML*
- *HTML*
- *CSS*

在 AJAX 中使用的开放标准被良好地定义，并得到所有主要浏览器的支持。AJAX 应用程序独立于浏览器和平台。（可以说，它是一种跨平台跨浏览器的技术）。

AJAX 事关更好的 Internet 应用程序

与桌面应用程序相比，Web 应用程序有很多优势：

- 可拥有更多用户
- 更容易安装和维护
- 更容易开发

但是， 应用程序不总是象传统应用程序那样强大和友好。

通过 AJAX，可以使 Internet 应用程序更加强大（更轻巧、更快速，且更易使用）。

今天您就可以开始使用 AJAX

没有什么新知识需要学习。

AJAX 基于开放的标准。而这些标准已被大多数开发者使用多年。

大多数 web 应用程序可通过使用 AJAX 技术进行重写，来替代传统的 HTML 表单。

AJAX 使用 XML 和 HTTP 请求

传统的 web 应用程序会把数据提交到 web 服务器（使用 HTML 表单）。在 web 服务器把数据处理完毕之后，会向用户返回一张完整的新网页。

由于每当用户提交输入，服务器就会返回新网页，传统的 web 应用程序往往运行缓慢，且越来越不友好。

通过 AJAX，web 应用程序无需重载网页，就可以发送并取回数据。完成这项工作，需要通过向服务器发送 HTTP 请求（在幕后），并通过当服务器返回数据时使用 JavaScript 仅仅修改网页的某部分。

一般使用 XML 作为接收服务器数据的格式，尽管可以使用任何格式，包括纯文本。

您将在本教程接下来的章节学习到如何完成这些工作。

PHP 和 AJAX

不存在什么 AJAX 服务器。

AJAX 是一种在浏览器运行的技术。它使用浏览器与 web 服务器之间的异步数据传输，使网页从服务器请求少量的信息，而不是整张页面。

AJAX 是一种独立于 web 服务器软件的 web 浏览器技术。

但是，在本教程中，我们将集中在运行在 PHP 服务器上的实际案例，而不是 AJAX 的工作原理。

如需阅读更多有关 AJAX 如何工作的知识，请访问我们的 AJAX 教程。

AJAX XMLHttpRequest

XMLHttpRequest 对象使 **AJAX** 成为可能。

XMLHttpRequest

XMLHttpRequest 对象是 AJAX 的关键。

该对象在 Internet Explorer 5.5 与 2000 年 7 月发布之后就已经可用了，但是在 2005 人们开始讨论 AJAX 和 Web 2.0 之前，这个对象并没有得到充分的认识。

创建 XMLHttpRequest 对象

不同的浏览器使用不同的方法来创建 *XMLHttpRequest* 对象。

Internet Explorer 使用 *ActiveXObject*。

其他浏览器使用名为 *XMLHttpRequest* 的 JavaScript 内建对象。

要克服这个问题，可以使用这段简单的代码：

```
var XMLHttp=null
if (window.XMLHttpRequest)
  {
  XMLHttp=new XMLHttpRequest()
  }
else if (window.ActiveXObject)
  {
  XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
```

代码解释：

1. 首先创建一个作为 XMLHttpRequest 对象使用的 *XMLHttp* 变量。把它的值设置为 null。
2. 然后测试 *window.XMLHttpRequest* 对象是否可用。在新版本的 Firefox, Mozilla, Opera 以及 Safari 浏览器中，该对象是可用的。
3. 如果可用，则用它创建一个新对象：*XMLHttp=new XMLHttpRequest()*
4. 如果不可用，则检测 *window.ActiveXObject* 是否可用。在 Internet Explorer version 5.5 及更高的版本中，该对象是可用的。
5. 如果可用，使用它来创建一个新对象：*XMLHttp=new ActiveXObject()*

改进的例子

一些程序员喜欢使用最新最快的版本的 XMLHttpRequest 对象。

下面的例子试图加载微软最新版本的 "Msxml2.XMLHTTP"，在 Internet Explorer 6 中可用，如果无法加载，则后退到 "Microsoft.XMLHTTP"，在 Internet Explorer 5.5 及其后版本中可用。

```
function GetXmlHttpObject()
{
var xmlHttp=null;

try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

代码解释：

1. 首先创建用作 XMLHttpRequest 对象的 *XMLHttp* 变量。把它的值设置为 null。
2. 按照 web 标准创建对象 (Mozilla, Opera 以及 Safari)：*XMLHttp=new XMLHttpRequest()*
3. 按照微软的方式创建对象，在 Internet Explorer 6 及更高的版本可用：*XMLHttp=new ActiveXObject("Msxml2.XMLHTTP")*
4. 如果捕获错误，则尝试更老的方法 (Internet Explorer 5.5) ：*XMLHttp=new ActiveXObject("Microsoft.XMLHTTP")*

更多有关 **XMLHttpRequest** 对象的信息

如果您希望阅读更多有关 XMLHttpRequest 的内容，请访问我们的 AJAX 教程。

PHP 和 AJAX 请求

- XMLHttpRequest

AJAX 请求

在下面的 AJAX 例子中，我们将演示当用户向 web 表单中输入数据时，网页如何与在线的 web 服务器进行通信。

这个例子包括三张页面：

- 一个简单的 HTML 表单
- 一段 JavaScript
- 一张 PHP 页面

HTML 表单

这是 HTML 表单。它包含一个简单的 HTML 表单和指向 JavaScript 的链接：

```
<html>
<head>
<script src="clienthint.js"></script>
</head>

<body>

<form>
First Name:
<input type="text" id="txt1"
onkeyup="showHint(this.value)">
</form>

<p>Suggestions: <span id="txtHint"></span></p>

</body>
</html>
```

例子解释 - HTML 表单

正如您看到的，上面的 HTML 页面含有一个简单的 HTML 表单，其中带有一个名为 "txt1" 的输入字段。

该表单是这样工作的：

1. 当用户在输入域中按下并松开按键时，会触发一个事件

2. 当该事件被触发时，执行名为 showHint() 的函数
3. 表单的下面是一个名为 "txtHint" 的 <span>。它用作 showHint() 函数所返回数据的占位符。

JavaScript

JavaScript 代码存储在 "clienthint.js" 文件中，它被链接到 HTML 文档：

```
var xmlHttp

function showHint(str)
{
if (str.length==0)
  {
  document.getElementById("txtHint").innerHTML=""
  return
  }
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
var url="gethint.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}

function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
```

```
  try
   {
   xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
   }
  catch (e)
   {
   xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
   }
  }
 return xmlHttp;
 }
```

例子解释：

showHint() 函数

每当在输入域中输入一个字符，该函数就会被执行一次。

如果文本框中有内容 (str.length > 0)，该函数这样执行：

1. 定义要发送到服务器的 URL（文件名）
2. 把带有输入域内容的参数 (q) 添加到这个 URL
3. 添加一个随机数，以防服务器使用缓存文件
4. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象，并在事件被触发时告知该对象执行名为 stateChanged 的函数
5. 用给定的 URL 来打开打开这个 XMLHTTP 对象
6. 向服务器发送 HTTP 请求

如果输入域为空，则函数简单地清空 txtHint 占位符的内容。

stateChanged() 函数

每当 XMLHTTP 对象的状态发生改变，则执行该函数。

在状态变成 4 （或 "complete"）时，用响应文本填充 txtHint 占位符 txtHint 的内容。

GetXmlHttpObject() 函数

AJAX 应用程序只能运行在完整支持 XML 的 web 浏览器中。

上面的代码调用了名为 GetXmlHttpObject() 的函数。

该函数的作用是解决为不同浏览器创建不同 XMLHTTP 对象的问题。

这一点在上一节中已经解释过了。

PHP 页面

被 JavaScript 代码调用的服务器页面是一个名为 "gethint.php" 的简单服务器页面。

"gethint.php" 中的代码会检查名字数组，然后向客户端返回对应的名字：

```
<?php
// Fill up array with names
$a[]="Anna";
$a[]="Brittany";
$a[]="Cinderella";
$a[]="Diana";
$a[]="Eva";
$a[]="Fiona";
$a[]="Gunda";
$a[]="Hege";
$a[]="Inga";
$a[]="Johanna";
$a[]="Kitty";
$a[]="Linda";
$a[]="Nina";
$a[]="Ophelia";
$a[]="Petunia";
$a[]="Amanda";
$a[]="Raquel";
$a[]="Cindy";
$a[]="Doris";
$a[]="Eve";
$a[]="Evita";
$a[]="Sunniva";
$a[]="Tove";
$a[]="Unni";
$a[]="Violet";
$a[]="Liza";
$a[]="Elizabeth";
$a[]="Ellen";
$a[]="Wenche";
$a[]="Vicky";

//get the q parameter from URL
$q=$_GET["q"];

//lookup all hints from array if length of q>0
if (strlen($q) > 0)
{
$hint="";
for($i=0; $i<count($a); $i++)
  {
  if (strtolower($q)==strtolower(substr($a[$i],0,strlen($q))))
    {
```

```
    if ($hint=="")
      {
      $hint=$a[$i];
      }
    else
      {
      $hint=$hint." , ".$a[$i];
      }
    }
  }
}

//Set output to "no suggestion" if no hint were found
//or to the correct values
if ($hint == "")
{
$response="no suggestion";
}
else
{
$response=$hint;
}

//output the response
echo $response;
?>
```

如果存在从 JavaScript 送来的文本 (strlen($q) > 0)，则：

1. 找到与 JavaScript 所传送的字符相匹配的名字
2. 如果找到多个名字，把所有名字包含在 response 字符串中
3. 如果没有找到匹配的名字，把 response 设置为 "no suggestion"
4. 如果找到一个或多个名字，把 response 设置为这些名字
5. 把 response 发送到 "txtHint" 占位符

6. XMLHttpRequest

PHP 和 AJAX XML 实例

AJAX 可与 **XML** 文件进行交互式通信。

AJAX XML 实例

在下面的 AJAX 实例中，我们将演示网页如何使用 AJAX 技术从 XML 文件中读取信息。

本例包括三张页面：

- 一个简单 HTML 表单
- 一个 XML 文件
- 一个 JavaScript 文件
- 一张 PHP 页面

HTML 表单

上面的例子包含了一张简单的 HTML 表单，以及指向 JavaScript 的链接：

```
<html>
<head>
<script src="selectcd.js"></script>
</head>

<body>

<form>
Select a CD:
<select name="cds" onchange="showCD(this.value)">
<option value="Bob Dylan">Bob Dylan</option>
<option value="Bee Gees">Bee Gees</option>
<option value="Cat Stevens">Cat Stevens</option>
</select>
</form>

<p>
<div id="txtHint"><b>CD info will be listed here.</b></div>
</p>

</body>
</html>
```

例子解释：

正如您看到的，它仅仅是一张简单的 HTML 表单，其中带有名为 "cds" 的下拉列表。

表单下面的段落包含了一个名为 "txtHint" 的 div。这个 div 用作从 web 服务器检索到的数据的占位符。

当用户选择数据时，会执行名为 "showCD" 的函数。这个函数的执行是由 "onchange" 事件触发的。

换句话说，每当用户改变了下拉列表中的值，就会调用 showCD 函数。

XML 文件

XML 文件是 "cd_catalog.xml"。该文件中包含了有关 CD 收藏的数据。

JavaScript

这是存储在 "selectcd.js" 文件中的 JavaScript 代码：

```
var xmlHttp

function showCD(str)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="getcd.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}

function GetXmlHttpObject()
{
var xmlHttp=null;

try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

例子解释：

stateChanged() 和 GetXmlHttpObject 函数与上一节中的相同，您可以参阅上一页中的相关解释。

showCD() 函数

假如选择了下拉列表中的某个项目，则函数执行：

1. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象
2. 定义发送到服务器的 URL（文件名）
3. 向 URL 添加带有下拉列表内容的参数 (q)
4. 添加一个随机数，以防服务器使用缓存的文件
5. 当触发事件时调用 stateChanged
6. 通过给定的 URL 打开 XMLHTTP 对象
7. 向服务器发送 HTTP 请求

PHP 页面

这个被 JavaScript 调用的服务器页面，是一个名为 "getcd.php" 的简单 PHP 文件。

这张页面是用 PHP 编写的，使用 XML DOM 来加载 XML 文档 "cd_catalog.xml"。

代码运行针对 XML 文件的查询，并以 HTML 返回结果：

```
<?php
$q=$_GET["q"];

$xmlDoc = new DOMDocument();
$xmlDoc->load("cd_catalog.xml");

$x=$xmlDoc->getElementsByTagName('ARTIST');

for ($i=0; $i<=$x->length-1; $i++)
{
//Process only element nodes
if ($x->item($i)->nodeType==1)
  {
  if ($x->item($i)->childNodes->item(0)->nodeValue == $q)
    {
    $y=($x->item($i)->parentNode);
    }
  }
}

$cd=($y->childNodes);

for ($i=0;$i<$cd->length;$i++)
{
//Process only element nodes
if ($cd->item($i)->nodeType==1)
  {
  echo($cd->item($i)->nodeName);
  echo(": ");
  echo($cd->item($i)->childNodes->item(0)->nodeValue);
  echo("<br />");
  }
}
?>
```

例子解释

当请求从 JavaScript 发送到 PHP 页面时，发生：

1. PHP 创建 "cd_catalog.xml" 文件的 XML DOM 对象
2. 循环所有 "artist" 元素 (nodetypes = 1)，查找与 JavaScript 所传数据向匹配的名字
3. 找到 CD 包含的正确 artist
4. 输出 album 的信息，并发送到 "txtHint" 占位符

PHP 和 AJAX MySQL 数据库实例

AJAX 可用来与数据库进行交互式通信。

AJAX 数据库实例

在下面的 AJAX 实例中，我们将演示网页如何使用 AJAX 技术从 MySQL 数据库中读取信息。

此列由四个元素组成：

- MySQL 数据库
- 简单的 HTML 表单
- JavaScript
- PHP 页面

数据库

将在本例中使用的数据库看起来类似这样：

id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

HTML 表单

上面的例子包含了一个简单的 HTML 表单，以及指向 JavaScript 的链接：

```
<html>
<head>
<script src="selectuser.js"></script>
</head>
<body>

<form>
Select a User:
<select name="users" onchange="showUser(this.value)">
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>

<p>
<div id="txtHint"><b>User info will be listed here.</b></div>
</p>

</body>
</html>
```

例子解释 - HTML 表单

正如您看到的，它仅仅是一个简单的 HTML 表单，其中带有名为 "users" 的下拉列表，这个列表包含了姓名，以及与数据库的 "id" 对应的选项值。

表单下面的段落包含了名为 "txtHint" 的 div。这个 div 用作从 web 服务器检索到的信息的占位符。

当用户选择数据时，执行名为 "showUser()" 的函数。该函数的执行由 "onchange" 事件触发。

换句话说：每当用户改变下拉列表中的值，就会调用 showUser() 函数。

JavaScript

这是存储在 "selectuser.js" 文件中的 JavaScript 代码：

```
var xmlHttp

function showUser(str)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="getuser.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("txtHint").innerHTML=xmlHttp.responseText
 }
}

function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 //Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

例子解释：

stateChanged() 和 GetXmlHttpObject 函数与 PHP AJAX 请求 那一节中的相同，您可以参阅其中的相关解释。

showUser() 函数

假如下拉列表中的项目被选择，函数执行：

1. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象
2. 定义发送到服务器的 URL（文件名）
3. 向 URL 添加带有下拉列表内容的参数 (q)
4. 添加一个随机数，以防服务器使用缓存的文件
5. 当触发事件时调用 stateChanged
6. 通过给定的 URL 打开 XMLHTTP 对象
7. 向服务器发送 HTTP 请求

PHP 页面

由 JavaScript 调用的服务器页面，是名为 "getuser.php" 的简单 PHP 文件。

该页面用 PHP 编写，并使用 MySQL 数据库。

其中的代码执行针对数据库的 SQL 查询，并以 HTML 表格返回结果：

```
<?php
$q=$_GET["q"];

$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
 {
 die('Could not connect: ' . mysql_error());
 }

mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = '".$q."'";

$result = mysql_query($sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";

while($row = mysql_fetch_array($result))
 {
 echo "<tr>";
 echo "<td>" . $row['FirstName'] . "</td>";
 echo "<td>" . $row['LastName'] . "</td>";
 echo "<td>" . $row['Age'] . "</td>";
 echo "<td>" . $row['Hometown'] . "</td>";
 echo "<td>" . $row['Job'] . "</td>";
 echo "</tr>";
 }
echo "</table>";

mysql_close($con);
?>
```

例子解释：

当查询从 JavaScript 被发送到这个 PHP 页面，会发生：

1. PHP 打开到达 MySQL 服务器的连接
2. 找到拥有指定姓名的 "user"
3. 创建表格，插入数据，然后将其发送到 "txtHint" 占位符

PHP 和 AJAX responseXML 实例

AJAX 可用于以 **XML** 返回数据库信息。

AJAX Database 转 XML 实例 （测试说明：该实例功能未实现）

在下面的 AJAX 实例中，我们将演示网页如何从 MySQL 数据库中读取信息，把数据转换为 XML 文档，并在不同的地方使用这个文档来显示信息。

本例与上一节中的 "PHP AJAX Database" 这个例子很相似，不过有一个很大的不同：在本例中，我们通过使用 responseXML 函数从 PHP 页面得到的是 XML 形式的数据。

把 XML 文档作为响应来接收，使我们有能力更新页面的多个位置，而不仅仅是接收一个 PHP 输出并显示出来。

在本例中，我们将使用从数据库接收到的信息来更新多个 <span> 元素。

此列由四个元素组成：

- MySQL 数据库
- 简单的 HTML 表单
- JavaScript
- PHP 页面

数据库

将在本例中使用的数据库看起来类似这样：

id	FirstName	LastName	Age	Hometown	Job
1	Peter	Griffin	41	Quahog	Brewery
2	Lois	Griffin	40	Newport	Piano Teacher
3	Joseph	Swanson	39	Quahog	Police Officer
4	Glenn	Quagmire	41	Quahog	Pilot

HTML 表单

上面的例子包含了一个简单的 HTML 表单，以及指向 JavaScript 的链接：

```
<html>
<head>
<script src="responsexml.js"></script>
</head>
<body>

<form>
Select a User:
<select name="users" onchange="showUser(this.value)">
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
<option value="3">Glenn Quagmire</option>
<option value="4">Joseph Swanson</option>
</select>
</form>

<h2>
<span id="firstname"></span> <span id="lastname"></span>
</h2>

<span id="job"></span>

<div style="text-align: right">
<span id="age_text"></span>
<span id="age"></span>
<span id="hometown_text"></span>
<span id="hometown"></span>
</div>

</body>
</html>
```

例子解释 - HTML 表单

- HTML 表单是一个下拉列表，其 name 属性的值是 "users"，可选项的值与数据库的 id 字段相对应
- 表单下面有几个 <span> 元素，它们用作我们所接收到的不同的值的占位符
- 当用户选择了具体的选项，函数 "showUser()" 就会执行。该函数的执行由 "onchange" 事件触发

换句话说，每当用户在下拉列表中改变了值，函数 showUser() 就会执行，并在指定的 <span> 元素中输出结果。

JavaScript

这是存储在文件 "responsexml.js" 中的 JavaScript 代码：

```
var xmlHttp
```

```
function showUser(str)
 {
 xmlHttp=GetXmlHttpObject()
 if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
 var url="responsexml.php"
 url=url+"?q="+str
 url=url+"&sid="+Math.random()
 xmlHttp.onreadystatechange=stateChanged
 xmlHttp.open("GET",url,true)
 xmlHttp.send(null)
 }

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
{
 xmlDoc=xmlHttp.responseXML;
 document.getElementById("firstname").innerHTML=
 xmlDoc.getElementsByTagName("firstname")[0].childNodes[0].nodeValu
 document.getElementById("lastname").innerHTML=
 xmlDoc.getElementsByTagName("lastname")[0].childNodes[0].nodeValue
 document.getElementById("job").innerHTML=
 xmlDoc.getElementsByTagName("job")[0].childNodes[0].nodeValue;
 document.getElementById("age_text").innerHTML="Age: ";
 document.getElementById("age").innerHTML=
 xmlDoc.getElementsByTagName("age")[0].childNodes[0].nodeValue;
 document.getElementById("hometown_text").innerHTML="<br/>From: ";
 document.getElementById("hometown").innerHTML=
 xmlDoc.getElementsByTagName("hometown")[0].childNodes[0].nodeValue
 }
}

function GetXmlHttpObject()
 {
 var objXMLHttp=null
 if (window.XMLHttpRequest)
  {
  objXMLHttp=new XMLHttpRequest()
  }
 else if (window.ActiveXObject)
  {
  objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
  }
 return objXMLHttp
 }
```

例子解释：

showUser() 与 GetXmlHttpObject 函数与 PHP 和 AJAX MySQL 数据库实例 这一节中的例子是相同的。您可以参阅其中的相关解释。

stateChanged() 函数

如果选择了下拉列表中的项目，该函数执行：

1. 通过使用 responseXML 函数，把 "xmlDoc" 变量定义为一个 XML 文档
2. 从这个 XML 文档中取回数据，把它们放在正确的 "span" 元素中

PHP 页面

这个由 JavaScript 调用的服务器页面，是一个名为 "responsexml.php" 的简单的 PHP 文件。

该页面由 PHP 编写，并使用 MySQL 数据库。

代码会运行一段针对数据库的 SQL 查询，并以 XML 文档返回结果：

```
<?php
header('Content-Type: text/xml');
header("Cache-Control: no-cache, must-revalidate");
//A date in the past
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");

$q=$_GET["q"];

$con = mysql_connect('localhost', 'peter', 'abc123');
if (!$con)
 {
 die('Could not connect: ' . mysql_error());
 }

mysql_select_db("ajax_demo", $con);

$sql="SELECT * FROM user WHERE id = ".$q."";

$result = mysql_query($sql);

echo '<?xml version="1.0" encoding="ISO-8859-1"?>
<person>';
while($row = mysql_fetch_array($result))
 {
 echo "<firstname>" . $row['FirstName'] . "</firstname>";
 echo "<lastname>" . $row['LastName'] . "</lastname>";
 echo "<age>" . $row['Age'] . "</age>";
 echo "<hometown>" . $row['Hometown'] . "</hometown>";
 echo "<job>" . $row['Job'] . "</job>";
 }
echo "</person>";

mysql_close($con);
?>
```

例子解释：

当查询从 JavaScript 送达 PHP 页面时，会发生：

- PHP 文档的 content-type 被设置为 "text/xml"
- PHP 文档被设置为 "no-cache"，以防止缓存
- 用 HTML 页面送来的数据设置 $q 变量
- PHP 打开与 MySQL 服务器的连接
- 找到带有指定 id 的 "user"
- 以 XML 文档输出数据

PHP 和 AJAX Live Search

AJAX 可为用户提供更友好、交互性更强的搜索体验。

AJAX Live Search

在下面的 AJAX 例子中，我们将演示一个实时的搜索。

实时的搜索与传统搜索相比，具有很多优势：

- 当键入数据时，就会显示出匹配的结果
- 当继续键入数据时，对结果进行过滤
- 如果结果太少，删除字符就可以获得更宽的范围

本例包括四个元素：

- 简单的 HTML 表单
- JavaScript
- PHP 页面
- XML 文档

在本例中，结果在一个 XML 文档 (links.xml) 中进行查找。为了让这个例子小而简单，我们只提供 8 个结果。

HTML 表单

这是 HTML 页面。它包含一个简单的 HTML 表单，针对此表单的 CSS 样式，以及指向 JavaScript 的链接：

```
<html>
<head>
<script src="livesearch.js"></script>
<style type="text/css">
#livesearch
  {
  margin:0px;
  width:194px;
  }
#txt1
  {
  margin:0px;
  }
</style>
</head>
<body>

<form>
<input type="text" id="txt1" size="30"
onkeyup="showResult(this.value)">

<div id="livesearch"></div>
</form>

</body>
</html>
```

例子解释 - HTML 表单

正如你看到的，HTML 页面包含一个简单的 HTML 表单，其中的文本框名为 "txt1"。

表单是这样工作的：

1. 当用户在文本框中按键并松开按键时，会触发一个事件
2. 当事件触发时，会执行名为 showResult() 的函数
3. 表单下面是名为 "livesearch" 的 <div> 元素。它用作 showResult() 所返回数据的占位符

JavaScript

JavaScript 代码存储在与 HTML 文档连接的 "livesearch.js" 中：

```
var xmlHttp

function showResult(str)
{
if (str.length==0)
```

```
 {
 document.getElementById("livesearch").
 innerHTML="";
 document.getElementById("livesearch").
 style.border="0px";
 return
 }

xmlHttp=GetXmlHttpObject()

if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }

var url="livesearch.php"
url=url+"?q="+str
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("livesearch").
 innerHTML=xmlHttp.responseText;
 document.getElementById("livesearch").
 style.border="1px solid #A5ACB2";
 }
}

function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
```

```
    }
  }
return xmlHttp;
}
```

例子解释：

GetXmlHttpObject 与 PHP 和 AJAX 请求 中的例子相同。

showResult() 函数

该函数每当一个字符输入文本框就会执行一次。

如果文本域中没有输入 (str.length == 0)，该函数把返回字段设置为空，并删除周围的任何边框。

不过，如果文本域中存在输入，则函数执行：

1. 定义发送到服务器的 url （文件名）
2. 把带有输入框内容的参数 (q) 添加到 url
3. 添加一个随机数，以防止服务器使用缓存文件
4. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象，并在触发一个变化时告知此函数执行名为 stateChanged 的一个函数
5. 使用给定的 url 来打开 XMLHTTP 对象
6. 向服务器发送 HTTP 请求

stateChanged() 函数

每当 XMLHTTP 对象的状态发生变化时，该函数就会执行。

当状态变为 4 (或 "complete") 时，就会使用响应文本来填充 txtHint 占位符的内容，并在返回字段周围设置一个边框。

PHP 页面

由 JavaScript 代码调用的服务器页面是名为 "livesearch.php" 的 PHP 文件。

"livesearch.php" 中的代码检查那个 XML 文档 "links.xml"。该文档 w3school.com.cn 上的一些页面的标题和 URL。

这些代码会搜索 XML 文件中匹配搜索字符串的标题，并以 HTML 返回结果：

```
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("links.xml");

$x=$xmlDoc->getElementsByTagName('link');
```

```
//get the q parameter from URL
$q=$_GET["q"];

//lookup all links from the xml file if length of q>0
if (strlen($q) > 0)
{
$hint="";
for($i=0; $i<($x->length); $i++)
 {
 $y=$x->item($i)->getElementsByTagName('title');
 $z=$x->item($i)->getElementsByTagName('url');
 if ($y->item(0)->nodeType==1)
  {
  //find a link matching the search text
  if (stristr($y->item(0)->childNodes->item(0)->nodeValue,$q))
   {
   if ($hint=="")
    {
    $hint="<a href='" .
    $z->item(0)->childNodes->item(0)->nodeValue .
    "' target='_blank'>" .
    $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
    }
   else
    {
    $hint=$hint . "<br /><a href='" .
    $z->item(0)->childNodes->item(0)->nodeValue .
    "' target='_blank'>" .
    $y->item(0)->childNodes->item(0)->nodeValue . "</a>";
    }
   }
  }
 }
}

// Set output to "no suggestion" if no hint were found
// or to the correct values
if ($hint == "")
 {
 $response="no suggestion";
 }
else
 {
 $response=$hint;
 }

//output the response
echo $response;
?>
```

例子解释：

如果从 JavaScript 送来了任何文本 (strlen($q) > 0)，会发生：

1. PHP 创建 "links.xml" 文件的一个 XML DOM 对象
2. 遍历所有 "title" 元素 (nodetypes = 1)，以便找到匹配 JavaScript 所传数据的 name
3. 找到包含正确 title 的 link，并设置为 "$response" 变量。如果找到多于一个匹配，所有的匹配都会添加到变量
4. 如果没有找到匹配，则把 $response 变量设置为 "no suggestion"
5. $result 是送往 "livesearch" 占位符的输出

PHP 和 AJAX RSS 阅读器

RSS 阅读器用于阅读 RSS Feed。

RSS 允许对新闻和更新进行快速浏览。

AJAX RSS 阅读器

在下面的 AJAX 实例中，我们将演示一个 RSS 阅读器，通过它，来自 RSS 的内容在不进行刷新的情况下载入网页。

本例包括三个元素：

- 简单的 HTML 表单
- JavaScript
- PHP 页面

HTML 表单

这是 HTML 页面。它包含一个简单的 HTML 表单和执行一个 JavaScript 文件的链接：

```
<html>
<head>
<script type="text/javascript" src="getrss.js"></script>
</head>
<body>

<form>
Select an RSS-Feed:
<select onchange="showRSS(this.value)">
<option value="Google">Google News</option>
<option value="MSNBC">MSNBC News</option>
</select>
</form>

<p><div id="rssOutput">
<b>RSS Feed will be listed here.</b></div></p>
</body>
</html>
```

例子解释 - HTML 表单

正如您看到的，上面的 HTML 页面包含一个简单的 HTML 表单，其中带有一个下拉列表框。

表单是这样工作的：

1. 当用户选择下拉框中的选项时，会触发一个事件
2. 当事件触发时，执行 showRSS() 函数

表单下面是名为 "rssOutput" 的一个 <div>。它用作 showRSS() 函数所返回的数据的占位符。

JavaScript

JavaScript 代码存储在 "getrss.js" 中，它与 HTML 文档相连接：

```
var xmlHttp

function showRSS(str)
 {
 xmlHttp=GetXmlHttpObject()
 if (xmlHttp==null)
  {
  alert ("Browser does not support HTTP Request")
  return
  }
 var url="getrss.php"
 url=url+"?q="+str
 url=url+"&sid="+Math.random()
 xmlHttp.onreadystatechange=stateChanged
 xmlHttp.open("GET",url,true)
 xmlHttp.send(null)
 }

function stateChanged()
 {
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
  {
  document.getElementById("rssOutput")
  .innerHTML=xmlHttp.responseText
  }
 }

function GetXmlHttpObject()
{
var xmlHttp=null;
try
 {
 // Firefox, Opera 8.0+, Safari
 xmlHttp=new XMLHttpRequest();
 }
catch (e)
 {
 // Internet Explorer
 try
  {
  xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
 catch (e)
  {
  xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
return xmlHttp;
}
```

例子解释：

stateChanged() 和 GetXmlHttpObject 函数与 PHP 和 AJAX 请求 这一节中的例子相同。

showRSS() 函数

每当在下拉框中选择选择时，该函数就会执行：

1. 定义发送到服务器的 url （文件名）
2. 把参数 (q) 添加到 url，参数内容是下拉框中的被选项
3. 添加一个随机数，以防止服务器缓存文件
4. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象，并告知该对象在触发一个改变时去执行 stateChanged 函数
5. 通过给定的 url 来打开 XMLHTTP
6. 把 HTTP 请求发动到服务器

PHP 页面

调用 JavaScript 代码的服务器页面是名为 "getrss.php" 的 PHP 文件：

```
<?php
//get the q parameter from URL
$q=$_GET["q"];

//find out which feed was selected
if($q=="Google")
 {
 $xml=("http://news.google.com/news?ned=us&topic=h&output=rss");
 }
elseif($q=="MSNBC")
 {
 $xml=("http://rss.msnbc.msn.com/id/3032091/device/rss/rss.xml");
 }

$xmlDoc = new DOMDocument();
$xmlDoc->load($xml);

//get elements from "<channel>"
$channel=$xmlDoc->getElementsByTagName('channel')->item(0);
$channel_title = $channel->getElementsByTagName('title')
->item(0)->childNodes->item(0)->nodeValue;
$channel_link = $channel->getElementsByTagName('link')
->item(0)->childNodes->item(0)->nodeValue;
$channel_desc = $channel->getElementsByTagName('description')
->item(0)->childNodes->item(0)->nodeValue;

//output elements from "<channel>"
echo("<p><a href='" . $channel_link
 . "'>" . $channel_title . "</a>");
echo("<br />");
echo($channel_desc . "</p>");

//get and output "<item>" elements
$x=$xmlDoc->getElementsByTagName('item');
for ($i=0; $i<=2; $i++)
 {
 $item_title=$x->item($i)->getElementsByTagName('title')
 ->item(0)->childNodes->item(0)->nodeValue;
 $item_link=$x->item($i)->getElementsByTagName('link')
 ->item(0)->childNodes->item(0)->nodeValue;
 $item_desc=$x->item($i)->getElementsByTagName('description')
 ->item(0)->childNodes->item(0)->nodeValue;

 echo ("<p><a href='" . $item_link
 . "'>" . $item_title . "</a>");
 echo ("<br />");
 echo ($item_desc . "</p>");
 }
?>
```

例子解释：

当一个选项从 JavaScript 发送时，会发生：

1. PHP 找出哪个 RSS feed 被选中
2. 为选中的 RSS feed 创建 XML DOM 对象
3. 找到并输出来自 RSS 频道的元素
4. 遍历前三个 RSS 项目中的元素，并进行输出

PHP 和 AJAX 投票

AJAX 投票

在这个 AJAX 实例中，我们将演示一个投票程序，网页在不重新加载的情况下，就可以获得结果。

本例包括四个元素：

- HTML 表单
- JavaScript
- PHP 页面
- 存放结果的文本文件

HTML 表单

这是 HTML 页面。它包含一个简单的 HTML 表单，以及一个与 JavaScript 文件的连接：

```
<html>
<head>
<script src="poll.js"></script>
</head>
<body>

<div id="poll">
<h2>Do you like PHP and AJAX so far?</h2>

<form>
Yes:
<input type="radio" name="vote"
value="0" onclick="getVote(this.value)">
<br />
No:
<input type="radio" name="vote"
value="1" onclick="getVote(this.value)">
</form>
</div>

</body>
</html>
```

例子解释 - HTML 表单

正如您看到的，上面的 HTML 页面包含一个简单的 HTML 表单，其中的 <div> 元素带有两个单选按钮。

表单这样工作：

- 当用户选择 "yes" 或 "no" 时，会触发一个事件
- 当事件触发时，执行 getVote() 函数
- 围绕该表单的是名为 "poll" 的 <div>。当数据从 getVote() 函数返回时，返回的数据会替代该表单。

文本文件

文本文件 (poll_result.txt) 中存储来自投票程序的数据。

它类似这样：

```
0||0
```

第一个数字表示 "Yes" 投票，第二个数字表示 "No" 投票。

注释：记得只允许您的 web 服务器来编辑该文本文件。不要让其他人获得访问权，除了 web 服务器 (PHP)。

JavaScript

JavaScript 代码存储在 "poll.js" 中，并于 HTML 文档相连接：

```
var xmlHttp

function getVote(int)
{
xmlHttp=GetXmlHttpObject()
if (xmlHttp==null)
 {
 alert ("Browser does not support HTTP Request")
 return
 }
var url="poll_vote.php"
url=url+"?vote="+int
url=url+"&sid="+Math.random()
xmlHttp.onreadystatechange=stateChanged
xmlHttp.open("GET",url,true)
xmlHttp.send(null)
}

function stateChanged()
{
 if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete")
 {
 document.getElementById("poll").
 innerHTML=xmlHttp.responseText;
 }
}

function GetXmlHttpObject()
{
var objXMLHttp=null
if (window.XMLHttpRequest)
 {
 objXMLHttp=new XMLHttpRequest()
 }
else if (window.ActiveXObject)
 {
 objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
 }
return objXMLHttp
}
```

例子解释：

stateChanged() 和 GetXmlHttpObject 函数与 PHP 和 AJAX 请求 这一节中的例子相同。

getVote() 函数

当用户在 HTML 表单中选择 "yes" 或 "no" 时，该函数就会执行。

1. 定义发送到服务器的 url （文件名）
2. 向 url 添加参数 (vote)，参数中带有输入字段的内容
3. 添加一个随机数，以防止服务器使用缓存的文件
4. 调用 GetXmlHttpObject 函数来创建 XMLHTTP 对象，并告知该对象当触发一个变化时执行 stateChanged 函数
5. 用给定的 url 来打开 XMLHTTP 对象
6. 向服务器发送 HTTP 请求

PHP 页面

由 JavaScript 代码调用的服务器页面是名为 "poll_vote.php" 的一个简单的 PHP 文件。

```
<?php
$vote = $_REQUEST['vote'];

//get content of textfile
$filename = "poll_result.txt";
$content = file($filename);

//put content in array
$array = explode("||", $content[0]);
$yes = $array[0];
$no = $array[1];

if ($vote == 0)
 {
 $yes = $yes + 1;
 }
if ($vote == 1)
 {
 $no = $no + 1;
 }

//insert votes to txt file
$insertvote = $yes."||".$no;
$fp = fopen($filename,"w");
fputs($fp,$insertvote);
fclose($fp);
?>

<h2>Result:</h2>
<table>
<tr>
<td>Yes:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($yes/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($yes/($no+$yes),2)); ?>%
</td>
</tr>
<tr>
<td>No:</td>
<td>
<img src="poll.gif"
width='<?php echo(100*round($no/($no+$yes),2)); ?>'
height='20'>
<?php echo(100*round($no/($no+$yes),2)); ?>%
</td>
</tr>
</table>
```

例子解释：

所选的值从 JavaScript 传来，然后会发生：

1. 获取 "poll_result.txt" 文件的内容
2. 把文件内容放入变量，并向被选变量累加 1
3. 把结果写入 "poll_result.txt" 文件
4. 输出图形化的投票结果

免责声明

W3School提供的内容仅用于培训。我们不保证内容的正确性。通过使用本站内容随之而来的风险与本站无关。W3School简体中文版的所有内容仅供测试，对任何法律问题及风险不承担任何责任。